2012 Special Issue

Metamodeling and the Critic-based approach to multi-level optimization

Ludmilla Werbos\textsuperscript{a,}\textsuperscript{*}, Robert Kozma\textsuperscript{b}, Rodrigo Silva-Lugo\textsuperscript{b}, Giovanni E. Pazienza\textsuperscript{b,1}, Paul J. Werbos\textsuperscript{c}

\textsuperscript{a}IntControl LLC and CLION, The University of Memphis, 38152 Memphis, TN, United States
\textsuperscript{b}CLION, The University of Memphis, United States
\textsuperscript{c}CLION and the National Science Foundation (NSF), United States

\begin{abstract}
Large-scale networks with hundreds of thousands of variables and constraints are becoming more and more common in logistics, communications, and distribution domains. Traditionally, the utility functions defined on such networks are optimized using some variation of Linear Programming, such as Mixed Integer Programming (MIP). Despite enormous progress both in hardware (multiprocessor systems and specialized processors) and software (Gurobi) we are reaching the limits of what these tools can handle in real time. Modern logistic problems, for example, call for expanding the problem both vertically (from one day up to several days) and horizontally (combining separate solution stages into an integrated model). The complexity of such integrated models calls for alternative methods of solution, such as Approximate Dynamic Programming (ADP), which provide a further increase in the performance necessary for the daily operation. In this paper, we present the theoretical basis and related experiments for solving the multistage decision problems based on the results obtained for shorter periods, as building blocks for the models and the solution, via Critic-Model-Action cycles, where various types of neural networks are combined with traditional MIP models in a unified optimization system. In this system architecture, fast and simple feed-forward networks are trained to reasonably initialize more complicated recurrent networks, which serve as approximators of the value function (Critic). The combination of interrelated neural networks and optimization modules allows for multiple queries for the same system, providing flexibility and optimizing performance for large-scale real-life problems. A MATLAB implementation of our solution procedure for a realistic set of data and constraints shows promising results, compared to the iterative MIP approach.
\end{abstract}

\section{Introduction}

This paper presents a combined method of solving and understanding scheduling and optimization tasks using both operations research tools and neural networks. For example, configuration and fleet assignment in realistic logistic networks is multidimensional with hundreds of thousands of variables and constraints, and it is computationally challenging. The best way to approach such problems at the present time is Mixed Integer Programming (MIP) (Dantzig & Thapa, 2003). This approach led to great success stories, such as (Barnhart & Schneuer, 1996; Ting & Tzeng, 2004), especially with the recent arrival of Gurobi software (Gurobi, 2011) and fast multicore computers. However, the more we succeed, the more we want. The scope of real-life problems is growing faster than software and hardware capabilities. A few years ago we would have been satisfied to be able to predict a typical day for manufacturing, distribution or logistics optimization purposes, but now we want to see an image of a typical weekend or a week. That would give the planners more power to consider all the opportunities and their combinations on a larger scale, instead of limiting themselves to a sequence of shorter periods of time modeled independently.

Efficient scheduling is a crucial issue in many areas belonging to both defense and civilian domains, including disaster response, power distribution, transportation, manufacturing job scheduling, etc. (Contesse et al., 2005; Kesen et al., 2010; Ko & Chang, 2008). Throughout the years, there have been countless efforts to tackle this problem; nevertheless, the appearance on the market of multicore platforms and advanced Field-Programmable Gate Arrays (FPGAs) offers unexplored possibilities, which require novel and more efficient approaches. Since the goal is to minimize the amount of time required to get to a desired level of accuracy, the best approach will include some use of metamodeling — the use of fast universal approximators such as neural networks to approximate slower, more expensive models or computations. The way such approximators are combined is determined by the principles of Adaptive Dynamic Programming (ADP) (Werbos, 2008).
Several possible configurations of the ADP-based system are discussed in this work. The paper is structured as follows: in Section 2, we present the principles of metamodeling; in Section 3, we introduce the representation of the optimization problem as a dynamic problem at multiple time scales; and in Section 4 we discuss the advantages and setbacks of various methods of ADP and their organization into a unified system, with respect to multi-period optimization. At the end of the paper, we draw conclusions and discuss the future developments of our project.

2. Principles of metamodeling

2.1. Forward metamodeling

In the world of stochastic optimization and operations research, “metamodeling” usually means forward metamodeling. In forward metamodeling, a neural network or some other function approximator is trained to approximate the function we are trying to minimize. We can represent a neural network as a function, \( f(\mathbf{x}, W) \), where \( \mathbf{x} \) are the inputs to the network, and \( W \) are the weights or parameters of the network. In the case of forwards metamodeling, \( f \) is just a scalar, so we may write the neural network as \( f(\mathbf{x}, W) \).

The task is to minimize the cost function \( C(\mathbf{u}, \alpha) \) subject to some constraints, where:

- \( C \) is the total cost,
- \( \alpha \) is the set of all the inputs, including those in the constraints,
- \( \mathbf{u} \) is the set of all the things we are trying to optimize, including the fleet assignments.

There are three steps to forward metamodeling here:

1. Pick a sample \( \{f_i\} \) of possible fleet assignments \( f_i \) — either all based on the same input vector \( \alpha \), or with different \( \alpha \). Let us assume that we have \( m \) sample points; in other words, \( i = 1 \) to \( m \).
2. Calculate \( C(f_i, \alpha) \) for each of these assignments.
3. Train the neural network \( f(\mathbf{x}, W) \) to approximate \( C \) over this sample; in other words, find the weights \( W \) which minimize the error between \( f \) and \( C \), for our sample of \( m \) cases, where \( \mathbf{x} \) (i) is set to \( \mathbf{u} \) or to \( \mathbf{u} \parallel \alpha \). Another way to say this is that we train the neutral network \( f \) over the set of sample pairs \( \{(\mathbf{u}, C(\mathbf{u}, \alpha))\} \).

Forward metamodeling is especially useful for stochastic optimization problems where it is very expensive to calculate the function \( C \), or where the only way to get its value is to do a physical experiment. But in the practical logistics problems we have been looking at, it is generally very easy to calculate the cost \( C \) after \( \alpha \) and \( \mathbf{u} \) are already known. Therefore, the usual forwards metamodeling is not so useful here.

2.2. Inverse metamodeling

The term “inverse metamodeling” is fairly new. The term itself originated in Barton and Meckesheimer (2006), but it can be generalized, to include the following kind of inverse metamodeling, described in Werbos (2010):

1. Pick a sample of possible \( \alpha_i \) \( \{\alpha_i\} \), with \( i = 1 \) to \( m \).
2. Run the optimizer to calculate the optimal \( \mathbf{u}, \mathbf{u^*}(\alpha_i) \), for each \( \alpha_i \) subject to the constraints.
3. Train the neural network \( f(\mathbf{x}, W) \) to approximate the function \( \mathbf{u^*}(\alpha_i) \) over the training set \( \{(\alpha_i, \mathbf{u^*}(\alpha_i))\} \). In other words, pick \( W \) so as to minimize the error between \( f \) and \( \mathbf{u^*} \), when \( \alpha_i \) is used as the input to the neural network.

If the neural network was a fast and exact approximator, this would be extremely useful in saving computation time; however, it is not likely to be quite so good in the beginning. It will require a very complex neural network to get a decent approximation in this application. This kind of research is the priority area for CLION, which specializes in complex neural networks. If the neural network is designed to match the capabilities of the dedicated hardware implementation, as much as possible, it should be very fast in real time. Even if the approximation is not perfect, this may provide a fast warm start for the optimizer program. The value of warm starts varies a lot from problem to problem, but generally provides a faster convergence compared to a random or randomized starting set of data.

2.3. Variations and extensions

1. Gradient-assisted learning (GAL). With forward metamodeling, we can usually calculate \( V_{\alpha}C \) for each sample point \( \mathbf{u} \), whenever we have a differentiable algorithm to calculate \( C \) itself, at a computational cost about the same as the computation of \( C \) itself. This comes from the use of the chain rule for ordered derivatives (Werbos, 2005).

   We can use Gradient Assisted Learning to train the neural network to match not only \( f \) and \( C \), but \( \nabla f \) and \( \nabla C \) as well. This can be extended to inverse metamodeling as well. If the vector \( \mathbf{u} \) has, say, 200 components, GAL causes a sample of \( N \) cases to be “worth” 200 times as much — as if \( 200 \times N \) samples had been collected. For inverse metamodeling of logistics tasks, it is necessary to get sensitivity information from the existing optimizer to make this possible, and build up new code.

2. Metaexploitation. This is the way of training a neural network to output a “good” new sample point, \( \mathbf{u}_{m+1} \) or \( \alpha_{m+1} \), for future use. In traditional stochastic search optimization, without metamodeling, the choice of a new sample point to explore is central to the power of the methods. In very complex systems, where an inverse metamodel can only provide a warm start, selection of a new point to follow up is also very important. In fact — the key idea in Werbos (2010) is that a layer of the cerebral cortex of the mammal brain performs metaexploration, and that this is what makes the mammal brain different from and more powerful than the reptile brain. We may or may not be ready to get into this kind of advanced capability, but it certainly is fundamental and important.

3. Robust optimization. In large scale logistics operations it is necessary to make decisions ranging from purchase commitments many years ahead, to actual day ahead fleet assignment. When the companies shift from long-term commitment to scheduling actual operations, they always have to make some (costly) adjustments, because of things which cannot be predicted ahead of time. Therefore, they can get better results by reformulating the optimization problem itself, by accounting for random disturbances. If one accounts for uncertainties in the vector \( \mathbf{u} \), the problem becomes more complicated in theory — but the actual surface of expected error becomes more convex, which would typically allow better use of new methods.

3. Problem representation

Consider a general case of a delivery company, which, for the purpose of our modeling, has overnight, two days, three days, and four days delivery service options. The company has a network of origins and destinations, the number of which runs in the order of hundreds, a dozen hubs, through which main cargo flows go, and a diverse fleet. For modeling purposes, we must account for various types of parcels, multiple delivery options, and multiple types of aircraft, restrictions on parking, size, overnight flying or
landing, and arrival or departure time restrictions in the airports. In addition, there might also be a specific holding pattern: a certain volume held at the origin or destination being allowed to accumulate according to service options. The operational goal of the company is to maximize the profit made on delivery of the maximally possible volume of parcels at minimal cost, subject to constraints. The Mixed Integer Linear Programming (MILP) model of this situation would involve hundreds of thousands of variables and constraints. It is not practical to attack such a problem head-on. We need to approach it iteratively, approximately, and ultimately—in a stochastic way.

When dealing with very complex problems, it is crucial to have an effective way to represent the knowledge and hence visualize information that would be otherwise very hard to retrieve from the raw data. Proper visualization helps in formulating the optimization problems and in assigning initial values and weights in a sequence of optimizers. For our experiments with data visualization, we used a network composed of sites located in the 48 contiguous US states chosen on the basis of their importance in the cargo market (World, 2009). We did not consider the sites with very low traffic among the sites used in the experiments, in order to avoid visual clutter. Each site was assigned some realistic volume of traffic, which we calculated from the available data of the last few years. Finally, we created a program in Wolfram Mathematica, which was used to generate the visual representations. Cargo volume can be interpreted as the number of pieces received/delivered at each location, but the representation is similar if we use other parameters, such as the weight of the freight or the number of vehicles employed.

A few key sites generate most of the volume. This observation is in accordance with the model of the freight traffic, as a scale-free network (Barabasi & Albert, 1999), in which a few nodes, called hubs, are connected to most of the remaining nodes and contribute a disproportionately large share of the traffic (Conway, 2004; DeLaurentis et al., 2004; Li & Cai, 2004). This phenomenon is represented in Fig. 1, in which we compare the cumulative percentage of the freight traffic (i.e., the sum of the traffic generated in n sites ordered by total, incoming and outgoing traffic), with a linear model, in which each site contributes 1/N to the total volume. Clearly, our model is highly nonlinear: the top 16% of the busiest sites account for 22% of the total traffic, the top 32% of the busiest sites account for 48% of the total traffic, and the top 45% of the busiest sites account for 66% of the total traffic. Modeling with 45% of the sites demonstrates the maximum discrepancy between the two models.

These considerations suggest to us that we can optimize the transportation among just a few sites thus obtaining a dramatic impact on the final result. In particular, in Fig. 1, one can see the network generated by the 7 busiest sites: now the network is simple enough to demonstrate the finest details of both the site locations and the connections among them, while still representing the 25% of the traffic. This will become a basis of the simplified model of the problem, suitable for comparison of various methods, in particular Neural Networks and Evolutionary Computing. We observed that an effective visualization method is a fundamental tool for representing the knowledge coming from the data, which especially in this kind of problem can be particularly cumbersome.

In a general case, ADP systems use Model-Critic-Action representation, which for our logistics problem is shown in Table 1.

The core methods of ADP (HDP, HDP, GDHP, etc.) were all first formulated as general methods (Werbos, 1981, 1987, 1992) for adapting the weights or parameters of these modules, giving the user great freedom in choosing the modules themselves. In our initial Case Study (Section 4.3), we have chosen to use the Gurobi optimizer itself as the Action module, while adapting a piecewise linear and piecewise quadratic Critic which Gurobi is capable of maximizing. In this Case Study, the state of the system at time t is described by the state vector X, which represents the types of the airplanes and the quantity of the airplanes of each type, at location L, at that time. The intrinsic value function U is the measure of profit already used in this application. We use the Critic to approximate the strategic utility J (sometimes called “the value function”) as a sum over the set of state variables, adding contributions from each variable separately, and the Action component is implemented via direct calls to the Gurobi MIP optimizer.

4. Adaptive dynamic programming for logistic optimization

Since our strength is Adaptive Dynamic Programming (ADP) (Werbos, 2004, 2012), if we reformulate the static optimization problem as a dynamic optimization problem at various time scales, we’ll be able to apply ADP methods, in a manner similar to Balakrishnan and Biega (1995), Balakrishnan et al. (2008) and Venayagamoorthy et al. (2003) for dynamic processes, as well as for transportation problems (Powell, 2007; Simao et al., 2009). For continuous variables, algorithmic implementation can be as basic as Backpropagation Through Time (BTT) (Werbos, 1994; Werbos et al., 1992). One can start with a goal for the week (to maximize total profit for the week), and then generate goals for each day to satisfy the weekly goal. The quality of each day solution is defined by its effect on the quality of solution for the subsequent days, guaranteeing overall system quality. However, BTT works
best with continuous variables, while most of the variables in the logistics situation are discrete. For these types of variables the methods such as Heuristic Dynamic Programming (HDP) (Werbos, 1992, 1977) and Globalized Dual Heuristic Programming (GDHP) (Werbos, 1981, 1987, 1992) are developed and patented by one of the authors (Werbos, 2003a, 2003b). If we look at the family of ADP designs, it is easy to see that Heuristic Dynamic Programming (HDP) is identical to Temporal Differences (Barto, 1992) for a discrete and continuous variables. ADP designs, it is easy to see that Heuristic Dynamic Programming (HDP) is identical to Temporal Differences (Barto, 1992) for a discrete set of choices. Dual heuristic programming (DHP) provides better performance than HDP when state variables are continuous, and Globalized DHP, GDHP, can be used when there is a mix of both discrete and continuous variables.

4.1. HDP as part of a logistic modeling system

As an intermediate step, to get near-term results, we are starting with the approach illustrated in Fig. 2. Our intention is to continue using the existing optimization software, such as Gurobi, but to enhance its usage so that the overall modeling system acquires a kind of look-ahead or foresight capability. No matter how complex the problems which can be solved today using such packages, we can simply handle the same degree of complexity better by using the existing optimization software to maximize something else besides the profit in the current period of time. For example, future profits will depend a lot on whether we position the fleet at the end of one stage of modeling to be better able to carry the cargo in the next day, and the day after that, for an entire week. By training a Critic network, we can add to utility function \( U \) an evaluation of the impact of that decision on future profit. The Critic would only give an approximation of those future benefits, but that is a big improvement over the status quo, in which the final position of the fleet is assumed to have no impact at all on later profits.

We can still use the best ADP methods and neural network methods for handling a complex network of coefficients (weights) in training that Critic network. For each time period \( t \), \( R(t) \) is where the fleet is in the beginning of the time period. Critic \( J(R(t), t, W) \) estimates Action (scheduling the fleet assignment \( u(t) \)) to maximize \( U(t) + J(t + 1) \). After the system calculates this maximum, it adjusts the weights \( W \) to make \( J(R(t), t, W) \) approximate \( U(t) + J(t + 1) \). This way we adapt our valuation of the present to better match our expectations about the future.

4.2. GDHP extension

Fig. 2 shows training of the Critic by Heuristic Dynamic Programming, as previously introduced in Werbos (1992). The summation of \( U(t) \) and \( J(t + 1) \) is performed inside the external optimizer, such as the Gurobi package we used in our numerical experiments. This is the modified Utility function, unlike the typical myopic approach, when only the quality of decision for the current stage is optimized. We train the coefficients of the Critic so that when it inputs \( R(t) \), it generates an output which is a better prediction of \( U(t) \) and \( J(t + 1) \). But the efficiency of training can be improved, even when using traditional optimizers, by using the GDHP method of training the Critic (Werbos, 1981, 1987), as illustrated in Fig. 3 for the general case.

In this application, each run of the linear or quadratic optimizer can be used to output not only \( U + J(t + 1) \), but its dual — its gradient with respect to all the variables processed by the Critic. Using an enhanced version of backpropagation, called Gradient Assisted Learning (Werbos, 2005) we can train the Critic so that its output and its gradient both match the target, \( J^* \). This improves the accuracy of training as if the size of the database were multiplied by the number of variables in the gradient; that can be extremely valuable, since the cost per run of the traditional optimizer system is relatively high.

In both HDP and GDHP

\[
J^*(R(t)) = U(R(t)) + \gamma J(R(t + 1))
\]  

(1)

where:

\( R(t) \) is the state of reality at time \( t \).

\( U(R(t)) \) is our ultimate evaluation (utility) of that state.

\( J(R(t)) \) is the Critic, our current approximation of the “value” of being in the state \( R(t) \). This value means the sum of its utility at present plus the sum of the utility of the future states it leads to. When we adapt the weights \( W \) of the Critic, we write it as \( \hat{J}(R(t), W) \).

\( J^*(R(t)) \) is the target we use at time \( t \), the (current) desired output, which we train the Critic to approximate.

In HDP, we train \( W \) so as to make \( \hat{J}(R(t), W) \) closer to \( J^*(R(t)) \). In GDHP, we train \( W \) so as to make \( \hat{J}(R(t), W) \) closer to \( J^*(R(t), W) \). GDHP can be implemented by using any supervised learning procedure to adjust \( W \), with \( R(t) \) the input vector and \( J^* \) the desired output. GDHP can be implemented using an augmented supervised learning procedure, using second-order backpropagation to minimize

\[
\alpha_0 (\hat{J}(R, W) - J^*(R))^2 + \sum_{i \in R} \alpha_i \left( \frac{\partial \hat{J}(R, W)}{\partial R_i} - \frac{\partial J^*(R)}{\partial R_i} \right)^2
\]  

(2)

with respect to the instances of \( W \) explicit in (2), where \( \alpha_i \) are adjustable learning parameters. This way we formulated a
consistent ADP algorithm for the multi-scale logistic optimization problem. In our initial case study (Section 4.3), we found it convenient to set $\alpha_0 = 0$ and $\alpha_i = 1$ for all $i$; in this special case, GDH reduces to a particularly simple form.

The main advantage of ADP approach for logistics problem lies in its lower computational complexity compared to Linear Programming (LP) methods. If allowed to run for unlimited time, LP methods will converge to a perfect solution, so they can not be beaten on quality. However, the simple Mixed Integer problem generates many linear or quadratic programming subproblems via branch-and-cuts. These subproblems are computationally expensive and require a significant amount of memory. Solving MIP for one operations day in our numerical experiments typically required 6–8 h of computational time, in optimal configuration of the optimizer on the multi-processor computer (Silva-Lugo et al., 2011). Details of parameter studies of a logistic model also can be found in the paper, which appeared in the IJCNN2011 proceedings (Werbos et al., 2011). The complexity of the problem grows cubically with time on real-life problems (Dantzig & Thapa, 2003), so forecast for a week ahead will be unfeasible using traditional methods. In case of ADP, at each stage of multi-stage problem we still need to solve the optimization problem, but the size of that problem is much smaller than complete week-ahead optimization, and the complexity of the multi-period optimization problem grows linearly with the number of periods.

Nonlinearity and unexpected changes in demand and other conditions are important in logistics, as they are in electric power and vehicle control and in other applications where adaptive dynamic programming (ADP) has been crucial or appears likely to become so (Werbos, 2011). To find the optimal solution for a multistage decision problem, subject to noise and nonlinearity, the most powerful general method is to use ADP with the most powerful neural network approximators used to supply at least the Critic—the approximator for the value function or its gradient. Because of the spatial complexity present in the logistics problem, and the possibility of using massively parallel new computer hardware, we would aim, in the long-term, to use a Cellular Simultaneous Recurrent Network with fast adaptive learning (Ilin et al., 2008) or an Object Network to serve as a Critic in these problems. Unfortunately, to reach that point, and maintain the degree of accuracy which large companies now expect in solving short-term optimization challenges here, we would need to replace the Action Network designs we have used in the past (Werbos, 1992) with a traditional linear or quadratic system, mostly using the simplex method, to optimize $U(t) + J(t + 1)$ for a complex nonlinear function $J$, as discussed in the following Section 4.3. The optimization can be done, in principle, by using nonlinear interior point methods (Werbos, 2012), which are also better suited to make full use of massively parallel new computer chips than currently used optimization methods, but that would require further research.

### 4.3. Case study for Piecewise-Linear Critic

In this example the state of the system at time $t$ is described by the state vector $X$, which represents the types of the airplanes and the quantity of the airplanes of each type, at location $L_i$ at that time. The Piecewise Linear Independent (PLI) method of approximating of strategic utility $J$ is used to model $J$ as a sum over the set of state variables, adding contributions from each variable separately.

$$j_{PLI}(x) = \sum_{a=1}^{T} \sum_{i=1}^{N} f_{ai}(x_{ai}, W^{[a,i]})$$

(3)

where $f_{ai}(x)$ are defined as a piecewise linear function of $x_{ai}$, with a set of adjustable parameters $W^{[a,i]}$.

The PLI Critic function is the sum over all combinations of airplane types and airport locations, of simple piecewise linear functions individually tuned via their sets of weights. The set of weights $W^{[a,i]}$ for each combination of location and airplane type consists of set slopes $[b]$, and set of inflection points $[c]$. Each such function is represented as

$$f(a_i, x_i) = b_1x_{ai} \text{ if } x_i < c_1$$
$$b_1C_i + b_2(x_{ai} - C_i) \text{ if } C_1 < x_{ai} < C_2$$
$$b_1C_i + b_2C_2 + b_3(x_{ai} - C_2) \text{ if } x_{ai} > C_2$$

(4)

To minimize costs over a one-week period we were using a MATLAB program called SimpleUpdate, which interfaced with Gurobi. We started with a set of locations of where the airplanes are at time zero, and with background exogenous information (EXO) at time zero. Each day was modeled independently. More precisely, for each day $t$, from day one ($t = 1$) to the last day ($t = 7$), we had a procedure which would input $x(t-1)$, the location of airplanes by type and by airport, and EXO($t-1$), the set of all other data (such as the amount of packages to be delivered to each destination) which are also important to the problem.

Our ultimate goal was as follows: Given $x(0)$, and exogenous variables EXO(0) through EXO(6), calculate fleet assignments for all seven days so as to maximize profit over all seven days. The results of the modeling were compared with the myopic “just the Gurobi, just for one day at a time” approach.

The traditional method, which was also implemented in MATLAB, for unification purposes, consisted of the following:

**SimpleUpdate($t$):**
- Input $x(t-1)$ and EXO($t-1$) (exogenous variables).
- Using Gurobi, pick fleet assignment for time $t$ which maximizes profit during time $t$, and calculate where that leaves airplanes at the end of the period (i.e. calculate the result $x(t)$). Main()
- Input $x(0)$ and EXO($t$) for $t = 0$ to 6.
- For $t = 1$ to 7.
- SimpleUpdate($t$).

Then add up total profit, and provide a report. The resulting $x(t)$ for $t = 1$ to 7 was saved for later use.

The first method we used for comparison (further referred to as Method 1) is called GDHPLL_update($t$), and represents an expanded version of SimpleUpdate($t$). It was expanded in two ways:

1. Instead of maximizing profit in time $t$, it maximized the sum of profit in time $t$ plus $\hat{J}(X(t))$, where $\hat{J}$ implements the piecewise linear $J$ function approximation we have discussed previously (see PLI). The GDHPLL_update module inputs the weights which are required for that approximator, as an argument, receiving them from the Main() program. Since the actual equations are all piecewise linear, they can be written as a Gurobi optimization instructions file.

2. GDH was used to update these weights. The fleet assignment has been calculated by Gurobi, in the expanded way, to maximize the function $ActualProfit = Profit + \hat{J}(X(t))$ was calculated for that fleet assignment. Then the weights $W$ in $\hat{J}$ were adapted so that the gradient of $\hat{J}(X(t-1))$ would be closer to the gradient of ActualProfit. Fortunately, the gradient of the objective function (ActualProfit) was available directly from Gurobi as the set of dual variables for $X(t-1)$.

This algorithm’s performance is now being compared with “myopic” approach of running Gurobi for seven days independently and totaling the values of the cost function. For the same computation time, the total value of the cost function is theoretically higher for the GDHPLL method. Since we used randomized
version of actual data in our simulation, the results are not presented here.

While presently we are passing the learning rate to GDHP-PLI_update as a parameter, in the future, the learning rate will be adapted using the methods previously developed in Ilin et al. (2008); Werbos (1992). Also, SimpleUpdate will need to be augmented to pass on additional output from Gurobi. The dual variables for \( X(t-1) \) form a vector \( \lambda(t-1) \), which allows us to use GDHP even in the beginning stages of this project. For each form of \( J \) that we plan to use, we need to tell Gurobi to maximize the ActualProfit with a corresponding \( J(X(1)) \). For each type of Critic; the subroutine GetActualProfit() would input \( X(t-1) \) and the weights in \( J(X(t)) \), and output \( P(T) \), profit in period \( T \), and the dual \( \lambda(t-1) \). For the forward pass, we just call that subroutine, and ignore the dual output, but then we can reuse the same subroutine for the backwards sweep to adapt the Critic, which is harder. It’s the same subroutine at each time, but the weights are different at each time. The forward pass would just access an internal database of what the Critic weights are for each time. For time \( T \), the Critic weights are all zero (no future past time \( T \), in the problem as stated.) The purpose of the backwards pass is to fill in that database of Critic weights, which the forwards pass will need.

For PLI, similarly to Powell (2007) we defined a cutoff \( C_i \) for each state variable \( x_i \) instead of adapting it. That resulted in

\[
\hat{J}(X) = \sum w_i x_i \text{if } x_i < C_i \\
\sum w_i x_i C_i + w_2 (x_i - C_i) \quad \text{if } x_i > C_i.
\]

(5)

In the special case of PLI the derivatives of \( J \) happen to have a very simple form: they are just the relevant weights for a particular interval of the input vector \( X \). As a result, PLI update using GDHP simply consists of adjusting each active weight to dual variable from Gurobi:

if \( x_i < C_i \)

\[ w_i = \begin{cases} w_i - (1 - \alpha) \text{old}_i & \text{if } x_i < C_i \\
\alpha \text{new}_i & \text{if } x_i > C_i \end{cases} \]

where \( \alpha \) is a learning rate.

Linear and Piecewise Linear Independent (PLI) methods are using that same subroutine. For linear, \( C_i \) is set to a value so high that the variable never actually reaches that. For PLI the cutoff for each variable \( x(i) \) is set up to equal the value of \( x(i) \) which we input from the starting values. That way the backward passes only need to estimate \( w_i \) for all \( i \) (in the linear case) and \( w_2 \) for all \( i \) (in the PLI case). Unlike in the approach of Powell (2007), here we are exploring the case of a MIP problem, not LP.

A more general implementation of GDHP for the piecewise linear Critic would be to add an intercept term \( J_0 \). At this stage we are not doing it, but for PLCP we will use an augmented Critic:

\[ \hat{J}_{PLCP}(X) = J_{PL}(X) + \sum a_i Q_{i}x_{1}x_{2} + Q_{i}x_{3} \]

This results in a quadratic form for ActualProfit, but Gurobi is now capable of solving MILP quadratic programming problems. As with PLI methods, the weights are updated using GDHP. Inclusion of the cross-terms and quadratics will offer a significant improvement over PLI, because interaction between different types of the aircraft at the same location has a direct impact on operations profit.

5. Conclusion and further development

The purpose of this study was to represent a multi-period logistics optimization problem as an Approximate Dynamic Programming problem, explicitly describing the Model, Critic, and Action components, and achieving a clear understanding of proper training algorithms for the networks involved. A case study for a limited set of locations and a particular type of Critic function was conducted, and demonstrated promising results. Current efforts include development of a version of the task more suitable for public release, and development of a modular MATLAB-based prototype for more advanced Critic functions and learning algorithms, which can be eventually implemented in high-performance software or hardware. In the case of real-life applications, ADP-based logistics optimization will make it possible to reduce the company costs (fleet size, inventory cost, maintenance cost, parking and landing/take-off fees, crew costs). This, in turn, will lead to the extension of the customer base by reducing the cut-off time for unit handling, and by steering the customer using price signals, possibly in real-time, towards service options that maximize overall delivery profit.

Acknowledgment

Partial support for this research from a FedEx Institute of Technology FIT Research Grant is gratefully acknowledged.

References


