Michael Sprengel, PhD
Engineer/Analyst
Czero

Introduction

Dynamic system modeling has revolutionized the way in which novel system architectures are designed and evaluated. Yet a system's design (architecture, component sizing, etc.) is only part of what determines a system's ultimate performance. How the system is controlled is often of equal importance. This is especially true with hybrid systems when power from multiple sources must be managed, though it applies to any system in which there are multiple state and control combinations which yield the desired output. Commonly a design team will be tasked with developing and employing a dynamic system model to both select a system architecture, and size components, with the goal of maximizing some performance metric. Too often the control system's design is relegated to second place: a low importance task which receives only the minimal effort and attention required to achieve a working system model.

Once to the point of comparing multiple system architectures/component sizing, this lack of focus on the control systems begs the question as to whether one system design outperforms another due to an inherently better configuration, or a more effective control strategy. There is certainly a risk that a poor system design with an effective control strategy could be chosen over a superior system design with an ineffective control strategy. While a control strategy may be improved and refined in time to be implemented in the final system, selection of a poor system design can be difficult to rectify as a project progresses. The use of an ineffective control scheme should not be viewed as negligence on part of the design team, rather it must be understood that developing a highly effective control scheme is often a challenging task, especially when the development concerns novel systems designs which are not well understood.

An effective solution to the problem of poor system control during the modeling and architecture selection phase is to apply a globally optimal controller directly to the dynamic simulation models. By using a globally optimal control scheme the influence of control on system performance can be eliminated, thereby enabling a fair system comparison. The most effective means of achieving optimal system control is through a technique known as dynamic programming. This white paper discusses the benefits of using dynamic programming as a design tool throughout a product's development, using a hybrid powertrain as an illustrative example.

Improving System Design and Performance Through Globally Optimal Control

Advantages of Dynamic Programming

Dynamic programming (DP) is powerful analytical tool which yields significant benefits when applied to dynamic simulation models. Some uses of DP include:

- Determining a system's best possible performance.
- Eliminating the influence of controls on system efficiency.
- Ensuring a fair and impartial comparison between system architectures and component sizing.
- Eliminating the risk of arriving at the wrong conclusion due to a poor system control.
- Discovering effective, but perhaps counterintuitive, control schemes.
- Developing control strategies based on the globally optimal results.
- Baselining implementable control strategies against the globally optimal solution.

Dynamic programming has one key benefit over other optimal control approaches:

- *Guarantees* a globally optimal state/control trajectory, down to the level the system is discretized to.

Dynamic programming also has several drawbacks which must be considered, including:

- Requires complete *a-priori* cycle knowledge (i.e. full knowledge of the upcoming cycle). As such DP is not an implementable control strategy.
- Is computationally expensive to solve as the required number of calculations increase exponentially with each additional state and control evaluated (curse of dimensionality). This disadvantage is somewhat mitigated by the highly parallelizable nature of DP.

Why do Effective Control Strategies Matter when Modeling?

High fidelity dynamic simulation models have become an essential and enabling tool in the development of innovative system architectures and technologies. Yet the dynamic model itself, which describes how a system's states will respond to a given set of control efforts, is only part of the story. Of equal importance is often how the system is controlled.

Take a power-split electric hybrid vehicle as an example. For a driver, instantaneous powertrain performance may be evaluated simply against its ability to provide the desired level of torque to the wheels when requested. Yet the system designers know that this requested torque may be achieved in numerous ways by varying the power split between the mechanical and electrical paths. From a strict instantaneous performance perspective, the choice of optimal power split is likely inconsequential. However, from an instantaneous efficiency perspective, the efficiency of each component (e.g. engine, battery, electric motors, etc.) within the powertrain must be considered holistically to determine the optimal power split. Figure 1 illustrates this point with a comparison between transmission efficiency and overall powertrain fuel consumption for an output-coupled power-split transmission operating under steady state conditions with no power being absorbed or supplied by the energy storage device (Sprengel, 2015).

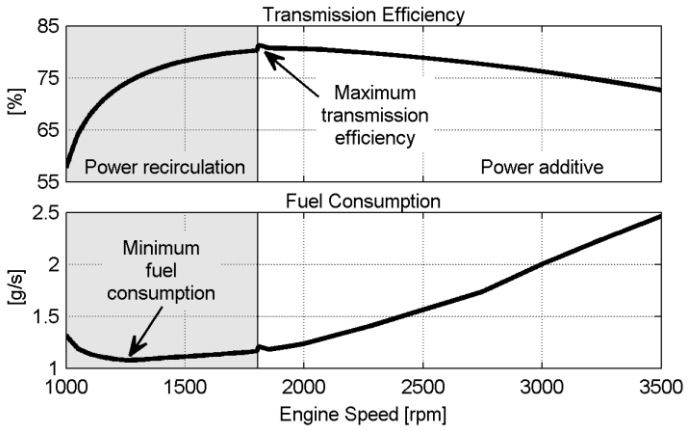

Figure 1: Powertrain vs. transmission efficiency

In this case it is more effective to operate the transmission in the less efficient power recirculation mode so that the overall powertrain operates in a more fuel-efficient manner.

While instantaneous optimization may be able to provide an optimal operating point for any given instance in time (as could be achieved by using Figure 1), this is often different from the globally optimal time domain solution. Returning to the electric power-split vehicle example, at a given point in time the instantaneously optimal control strategy from a fuel economy perspective may be to meet the required torque by maximizing the power discharged from the battery. However, such a control strategy could deplete the battery before the vehicle arrives at its destination, causing the powertrain to rely solely on the engine for the remainder of the trip. This would almost certainly decrease the overall cycle efficiency relative to what could have been achieved if the powertrain were more effectively controlled. The system designers must balance not only the power split between the mechanical and electric paths to achieve the highest possible powertrain efficiency, but must also balance the charging and discharging of the battery to maximize overall cycle efficiency.

These efforts are further complicated by the need to account for and control the system relative to events which have not yet occurred. For example, there will likely be different optimal control strategies for a vehicle which is about to drive up a long hill, versus one which is about to come to a rapid stop. To maximize cycle efficiency the powertrain must begin preparing for each of these events before they occur (e.g. storing extra energy before driving up the hill, or discharging energy before braking to ensure adequate capacity exists to capture and store the vehicle's kinetic energy). The use of complete *a-priori* cycle knowledge is a key to DP's ability to determine and guarantee globally optimal solutions as will be discussed in the next section.

To further illustrate the influence of control on system performance, Table 1 show a comparison between two power management controllers relative to the globally optimal solution for a plug-in electric hybrid SUV achieved through DP (Gong et al., 2008).

Table 1: Fuel economy comparison (Gong et al., 2008)

Drive Cycle	UDDS[1]	US06[2]	ECE-EUDC[3]	HW-FET[4]
Control Strategy	*Fuel Economy [l/100 km]*			
DP Charge Depletion	4.27	4.47	3.76	2.88
Depletion Sustenance	6.10	9.80	6.30	5.70
Rule-Based	4.30	10.50	7.80	7.90
	Percent Decrease in Fuel Economy Relative to DP [%]			
DP Charge Depletion	-	-	-	-
Depletion Sustenance	42.9	119.2	67.6	97.9
Rule-Based	0.7	134.9	107.4	174.3

1. Urban Dynamometer Driving Schedule
2. US06 Supplemental Federal Test Procedure
3. Economic Commission for Europe – Extra Urban Driving Cycle
4. Highway Fuel Economy Test

Clearly apparent here are the significant differences between the globally optimal DP strategy, and the other two control strategies. It is also worth noting that for the UDDS cycle the rule-based control scheme achieved results very close to the globally optimal solution. This demonstrates that implementable controllers, when well designed, can also achieve near optimal performance. Yet when this same rule-based control scheme was evaluated on cycles other than the UDDS cycle, it achieved significantly worse results highlighting its limitations. In fact, one could imagine that if such a hybrid vehicle were controlled poorly, that the resulting fuel economy could be worse than a convention non-hybrid vehicle. This could lead to an incorrect perception of the technology when in fact the control scheme, not the base system architecture, was causing the poor system performance.

The purpose of this discussion on power management for hybrid vehicles is not intended to explore the pros and cons of various control strategies, but rather to illustrate the complexity of such control and how ineffective control schemes may provide an inaccurate perception of a system during the modeling and architecture selection phase. While this paper illustrates DP's utility for a hybrid vehicle power management, optimal control through DP can be applied to many other applications as well. Ultimately any system whose states and controls are not deterministic (i.e. fully defined) may benefit from DP.

How Dynamic Programming Works

Dynamic programming is based Richard Bellman's Principle of Optimality: "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." (Bellman, 1957) Rephrased: Optimization of the future does not depend on what occurred in the past. This basic principle has been extended and implemented in a wide variety of optimal control algorithms. Many of these approaches rely on solving the Hamilton–Jacobi–Bellman partial differential equation for continuous time systems. However, using partial differential equations often requires a rigid formulation of the state space equations which largely precludes the use of lookup tables, non-continuous functions, and other inequalities and constraints which are commonly used in dynamic simulation models. Another class of DP algorithms discretize the system and implement a numerical approach to solve the optimal control problem. It is this discretized DP formulation which is well suited for the class of dynamic system models discussed in this work. The basic operation of a discretized DP algorithm can be explained by first applying it to a multistage decision-making process. Bellman's principle of optimality is explained graphically through Figure 2.

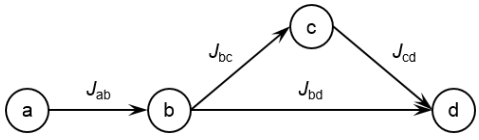

Figure 2: Multistage decision process

In this example J represents the cost of transition between discrete states *a*, *b*, *c*, and *d*. Here two possible paths exist between states *a* and *d*: *a*-*b*-*c*-*d* and *a*-*b*-*d* with the optimal path (denoted by *) defined as:

$$J_{ad}^{*} = J_{ab} + \min(J_{bcd} + J_{bd})$$

Applying the principle of optimality yields the assertion that if *b*-*c*-*d* is the optimal path between *b*-*d* then a-*b*-*c*-*d* is the optimal path between *a*-*d*. While this may

appear to be a trivial solution, when applied to larger and more complex problems it has powerful ramifications.

Bellman formed a computational method (algorithm) known as DP by extending his principle of optimality to a sequence of decisions. By using the concept that wherever an optimization is begun, the remaining path must be optimal, Bellman subdivided complex multistage decision problems into a series of simpler one stage sub problems. Beginning with the final state, Bellman worked backwards through the decision process to obtain the globally optimal solution with a computational expense far less than direct enumeration. This process can be explained graphically using Figure 3. Here the transitional costs between each state have been given numerical values as denoted on the figure.

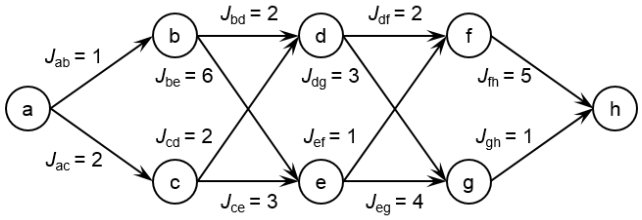

Figure 3: Multistage DP example, step 1

Using direct enumeration requires the evaluation of all 8 possible paths between *a* and *h*:

$$J_{ah}^{*} = min \begin{pmatrix} J_{ab} + J_{bd} + J_{df} + J_{fh}, & J_{ac} + J_{cd} + J_{df} + J_{fh} \\ J_{ab} + J_{bd} + J_{dg} + J_{gh}, & J_{ac} + J_{cd} + J_{dg} + J_{gh} \\ J_{ab} + J_{be} + J_{ef} + J_{fh}, & J_{ac} + J_{ce} + J_{ef} + J_{fh} \\ J_{ab} + J_{be} + J_{eg} + J_{gh}, & J_{ac} + J_{ce} + J_{eg} + J_{gh} \end{pmatrix}$$

DP takes a different approach and subdivides the problem into multiple stages (Figure 4). The DP algorithm begins one stage before the final stage (Stage 4) and records the optimal transitional cost *J** between each state in the current stage, and the final stage (Stage 5) (a trivial process at this point as only one possible decision exists for each state).

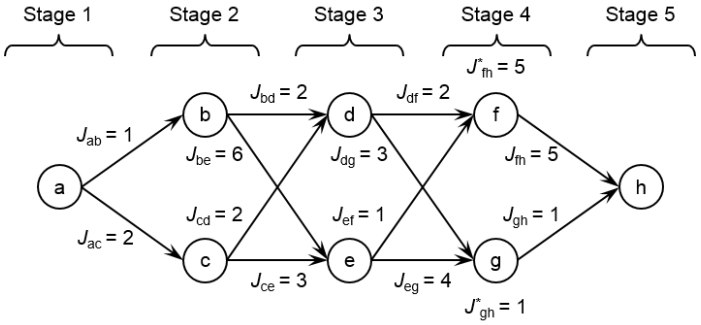

Figure 4: Multistage DP example, step 2

Next the DP algorithm steps back one stage (to Stage 3) and repeats the optimization process. However instead of optimizing the path all the way from Stage 3 to Stage 5, the DP algorithm has only to optimize between Stage 3 and Stage 4 as the optimal path from Stage 4 to Stage 5 has already been determined and recorded. The cost which is recorded at each state in Stage 3 contains not only the transitional cost between Stage 3 and Stage 4, but also the cost to finish from Stage 4 to the final stage. This cumulative cost to finish is one of the key concepts of DP and is known as the embedding principle. It is this running tally of the optimal cost to finish which enables the multistage decision process to be subdivided into a series of one stage sub problems. This process of optimizing each stage in turn repeats recursively until the initial decision is reached (Figure 5).

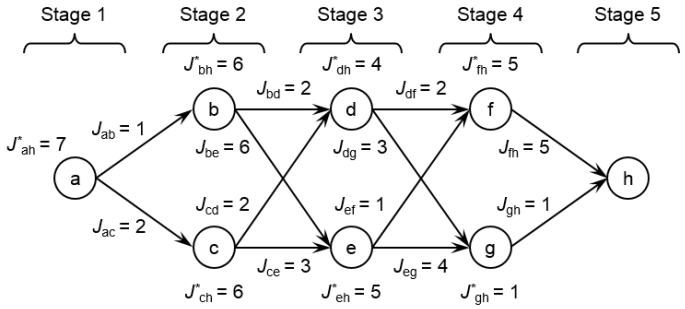

Figure 5: Multistage DP example, step 3

Once the initial state is reached, the backwards stepping component of the DP algorithm is concluded. In order to extract the optimal decision sequence, one needs merely to follow the optimal decisions forward from state to state. DP tabulates not only the optimal decision path from the first state to the last, but also the optimal decision path from any state to the final state. Bellman's principle of optimality provides a sufficient condition for optimality. That is because all possible

optimal candidate decision paths are analyzed, the optimal path found must be the globally optimal path. More insight into global optimality can be found by once more investigating the graphical example. Now a forward stepping instantaneous optimization algorithm has been used to find the instantaneously optimal path from state *a* to *h*. At each state this algorithm chooses the decision with the lowest instantaneous cost yielding the dashed path shown in Figure 6.

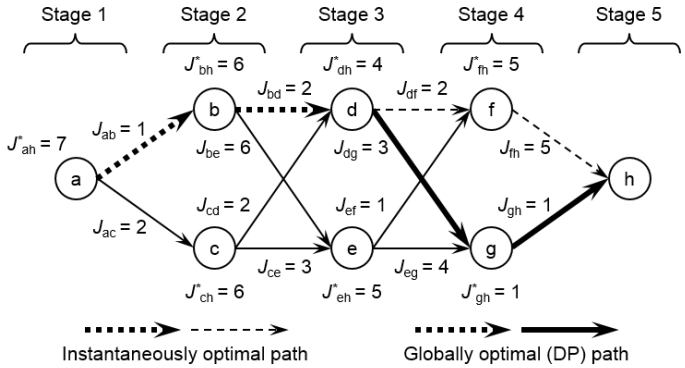

Figure 6: Multistage DP example, step 4

Here the instantaneous optimization algorithm followed the path *a-b-d-f-h* with a cost of 10, while the globally optimal path *a-b-d-g-h* found by DP (bold lines) resulted in a cost of 7. Of interest is the path taken by both algorithms from Stage 3 to Stage 4. Here both algorithms began on State *d,* but while the instantaneous optimal path *d-f* follows the lowest cost decision from Stage 3 to Stage 4, the globally optimal path *d-g* follows a trajectory with a higher decision cost. This example illustrates that a globally optimal path may require locally suboptimal decisions to yield the minimum overall cost.

DP's predominant advantage over direct enumeration is its substantial reduction in computational expense. Table 2 compares the computation expense of both methods assuming S=10 states and C=5 controls. Even though the number of stages, states, and controls are at least an order of magnitude smaller than the values commonly required in DP evaluations of dynamic systems, the computational expense of direct enumeration is already infeasible.

Table 2: Comparison of computational expense between DP and direct enumeration (for 10 states and 5 controls)

Number of stages	Calculations required by DP	Calculations required by direct enumeration
1	50	50
10	500	122,070,300
25	1,250	3,725,290,298,461,914,050
N	$S \cdot C \cdot N$	$\sum_{K=1}^{N} [S \cdot C^K]$

Applying Dynamic Programming to Discrete Time Optimal Control

Thus far a general overview of DP has been given. This process will now be expanded upon for the specific case of discrete time optimal control. Discrete time DP requires a system to be expressed using a state space representation. In the state space system key system parameters and controls are represented by state variables $x_i(t)$, and control inputs $u_i(t)$, respectively. The state variables represent the minimum number of parameters which must be known to fully describe a system at any point in time, while control inputs refer to system inputs which serve to alter these states. Together the set of all states and controls for a system are expressed through state vector $X(t)$ and control vector $U(t)$.

$$X(t) \equiv [x_1(t)\ x_2(t)\ \cdots\ x_n(t)\ ]^T$$
$$U(t) \equiv [u_1(t)\ u_2(t)\ \cdots\ u_n(t)\ ]^T$$

When the state and controls are related by 1st order differential equations, a nonlinear time varying physical system can be described by the state equation:

$$\dot{x}(t) = a(x(t), u(t), t)$$

However, it is also permissible to describe the system through any type of model (e.g. lumped parameter, distributed parameter, black box, etc.) as long as the value of appropriate states and controls can be extracted and applied at the necessary points in time.

DP requires both the continuous time and states of a continuous time optimal control problem to be

discretized. The discretized time can be equated to the stages in the aforementioned graphical example, while the system's discretized states represent the discrete states within each stage. DP does not require controls to be discretized, however doing so generally improves computational efficiency for certain classes of problems.

The principle of optimally is expressed mathematically for discrete time dynamic programming through the functional recurrence equation of DP:

$$J^*_{N-K,N}\big(x(N-K)\big) = \min_{u(N-K)} \left\{ \overbrace{g_D\left(a_D\big(x(N-K),u(N-K)\big)\right)}^{Transitional\ Cost} + \overbrace{J^*_{N-(K-1),N}\left(a_D\big(x(N-K),u(N-K)\big)\right)}^{Cost\ to\ Finish} \right\}$$

Where N is the number of stages, K is the stage counter, g_D is the transitional cost function to be minimized between the current and subsequent state, and a_D represents the system dynamics.

To better illustrate the application of DP to state/control trajectory optimization, a series hydraulic hybrid transmission (Figure 7) will now serve as a reference system.

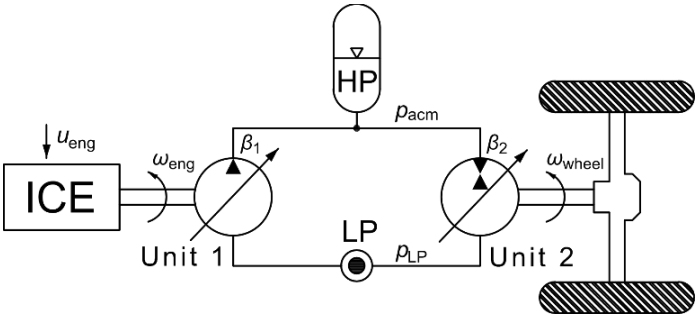

Figure 7: Series hydraulic hybrid

Appling DP to the series hybrid begins by forming the state and control vectors:

$$X \equiv \begin{bmatrix} \omega_{eng} & \omega_{wheel} & P_{acm} & P_{LP} \end{bmatrix}^T \quad U \equiv \begin{bmatrix} u_{eng} & \beta_1 & \beta_2 \end{bmatrix}^T$$

Where ω_{eng} is the engine speed, ω_{wheel} is the wheel speed, p_{acm} is the pressure of the high-pressure accumulator, p_{LP} is the pressure of the low-pressure system, u_{eng} is the engine throttle, and β_1, β_2 are the displacements of Units 1 and 2 respectively.

As the computational expense of DP grows exponentially with the addition of each state and control, it is highly desirable to eliminate superfluous states and controls whenever possible. In this example the powertrain will be optimized over a predefined drive cycle. As such the wheel speed ω_{wheel} is known as a function of time and can thus be eliminated from the state vector. The required wheel torque can also be determined as a function of time using the predefined drive cycle and a vehicle dynamics model. Consequently Unit 2's displacement β_2 becomes a function of time and accumulator pressure and is likewise removed from the control vector. Finally, to reduce computational expense, it is assumed that the low-pressure system p_{LP} maintains a constant set pressure. These simplifications result in the following reduced state space vectors:

$$X \equiv \begin{bmatrix} \omega_{eng} & P_{acm} \end{bmatrix}^T \quad U \equiv \begin{bmatrix} u_{eng} & \beta_1 \end{bmatrix}^T$$

Configuring the system for DP continues with determining how to discretize the continuous states and time (i.e. split continuous variables into the descrete set of values requried by DP). This is an important step as the level of discretization has a direct impact on both solution accuracy (which roughly converges asymptotically as discretization increases) and computational expense (which increases linearly as discretization increases). Thus close attention must be paid to the level of discretization in order to balance acceptable solution accuracy with feasible computational expense.

When discretizing states the concept of accessibility is useful. Accessibility refers to the ability of one state to transition to other states within a single time step. A state with low accessibility may only be capable of transitioning to a few different states (or maybe none at all) within a given time step. Whereas a state with high accessibility would be able to reach many different states within the same time frame. Using the series hydraulic hybrid as an example, the maximum change in system pressure during a single DP time step is dicated by system dnyamics. The graularity of the system's pressure discritization must therefore be sufficently fine as to allow the system pressure to move

between states within a single DP time step. Failure to sufficently discritizte system pressure could result in the DP algorithm falsly indicating that a constant system pressure was optimal, when in acutality this finding was a result of poor state discritzation. An example of accessibility is given in Figure 8.

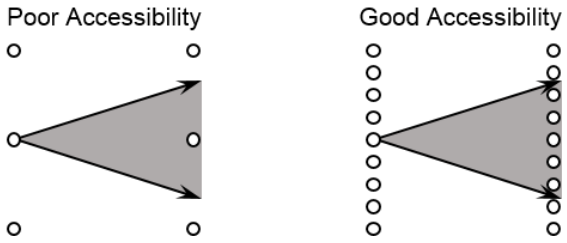

Figure 8: Dynamic programming accessibility

In general, states which change slowly required a higher level of discretization than states which change quickly. For the series hybrid example the following state discretizations are reasonable:

$$X \equiv \begin{bmatrix} \omega_{eng} \\ P_{acm} \end{bmatrix} \begin{bmatrix} 750 - 4000\ rpm \\ 145 - 370\ bar \end{bmatrix}$$
$$\begin{bmatrix} 750{:}\,25{:}\,1000, 1050{:}\,50{:}\,1500, 1600{:}\,100{:}\,4000 \\ 145{:}\,5{:}\,370 \end{bmatrix}$$

Note the engine speed has a non-uniform discretization. This enables the designer to achieve greater accuracy in areas which are of greater interest (e.g. low engine speed) while reducing computational expense in areas which are unlikely to see much operation (e.g. high engine speed).

Another factor which must be determined is the degree to which time is discretized. While a finer time discretization will improve solution accuracy, it will also limit how much a given state can change within a time step (thus requiring an even finer state discretization). Each discrete time step becomes a stage within the DP algorithm where controls are optimized and held constant for the entire time step. This does not mean the model (equations) must be solved using the DP time step. Rather any time step may be used for the model solvers so long as the model is simulated for the duration of the DP time step.

An example of a backwards stepping time discretization is shown in Figure 9. Note the time discretization shown in this figure is highly exaggerated to demonstrate the concept; the time discretization should actually be substantially smaller to accurately capture the drive cycle dynamics.

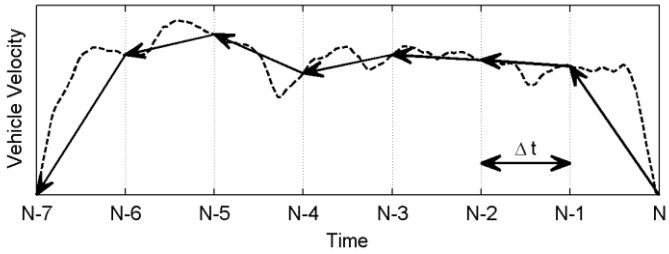

Figure 9: Dynamic programming time discretization

For powertrain optimization a one second DP time step is generally appropriate due to the system's relatively slow dynamics. The example series hybrid transmission will be optimized over the industry standard UDDS cycle. This 1369 second long drive cycle (Figure 10) is indicative of urban driving and well suited for evaluating hybrid powertrains.

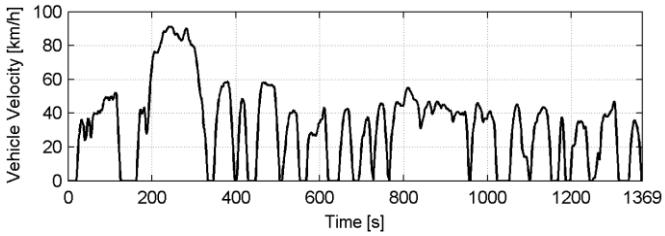

Figure 10: Urban Dynamometer Driving Schedule

With the state and time discretizations finalized, several matrices must now be initialized. These include the optimal cost matrix J^*, and two optimal control matrices U^* (one for each state). Note the dimensions of the matrices correspond to the discretization of the state and time vectors (t:1370, ω_{eng}:46, P_{acm}:46).

$$J^*_{N,x_1,x_2} = J^*_{1370,46,46} = [\,]$$
$$U^*_{(1)N,x_1,x_2} = U^*_{(1)1370,46,46} = [\,]$$
$$U^*_{(2)N,x_1,x_2} = U^*_{(2)1370,46,46} = [\,]$$

DP optimizes the controls for each state at every DP time step. While many different optimization techniques would be valid for solving a single step of the functional recurrence equation, a full factorial search of discretized controls has proven to be the fastest approach for the DP algorithm used in this work. Controls are discretized using similar principles as the states. However, states which change fast (i.e. a stiff system) generally need finer control discretization than

states which change slowly to enable accurate optimization. Engine dynamics are a good example of a stiff system: even small discrepancies between the combustion and load torques will result in a significant change in engine speed over the DP time step. Such a system would require a very high level of discretization on the throttle to enable a constant engine speed to be maintained in the face of varying loads. An alternative used in this work is to replace the throttle control with a desired engine speed $\omega_{\text{eng\_des}}$. The enables a controller within the simulation model to continuously control the engine throttle to track the reference engine speed, all while requiring less control discretization. Together the controls were discretized as follows:

$$U \equiv \begin{bmatrix} \omega_{eng\_des} \\ \beta_1 \end{bmatrix} \begin{bmatrix} 750 - 4000\ rpm \\ 0 - 100\% \end{bmatrix}$$
$$\begin{bmatrix} 750{:}\,25{:}\,1000, 1050{:}\,50{:}\,1500, 1600{:}\,100{:}\,4000 \\ 0{:}\,5{:}\,100 \end{bmatrix}$$

With the transmission model constructed, states and controls identified and discretized, and the performance metric specified (minimizing fuel consumption), the DP algorithm begins one time step before the final time step. The model is then initialized at each combination of discrete states $\left(\omega_{\text{eng}}(46)\ x\ P_{\text{acm}}(46) = 2116\right)$ before every combination of discrete controls is applied in turn $\left(\omega_{\text{eng\_des}}(46)\ x\ \beta_1(21) = 966\right)$ and the system is simulated forward in time for one DP time step.

For this stage (N-1) the total cost being minimized is simply the transitional fuel consumption between the current stage and the final stage. After all sets of controls are applied for a specific state, the DP algorithm selects and records the minimum cost for that specific state in the J^* matrix. The associated optimal control values $\left(\omega_{\text{eng\_des}}, \beta_1\right)$, which resulted in this minimum cost, are also recorded in their respective optimal control matrices U^* for the state which was just evaluated. This projection of states by means of various controls is illustrated in Figure 11.

The optimal controls are those which minimize both the transitional fuel consumption, and the cost to reach the end of the cycle. (While at stage N-1 the transitional fuel consumption, and the cost to reach the end of the cycle, are one in the same. In subsequent time steps (N-2, N-3, …) the transitional fuel consumption will be determined based on where the project state lands on the cost to finish matrix J^*).

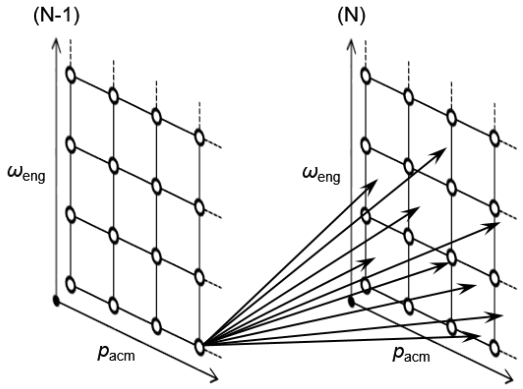

Figure 11: Dynamic programming state projection

Once all the states have been optimized, the DP algorithm steps back in time to stage (N-2) and begins again. Once again the model is initialized at a given state and controls are applied and simulated for a single time step. However now the resulting states after the DP time step (now at stage N-1) are used to determine the cost to finish by referencing the optimal cost to finish from time step N-1 to the final time step contained within the J^* matrix.

This recursive process is repeated from stage to stage until the initial time step is reached, thus concluding the backwards stepping portion of the DP algorithm. Once the initial time step is reached both the J^* and U^* matrices are filled. However, the J^* matrix is of little use as the cost to finish values provide little information regarding the optimal path through the states. The only information which can be extracted from the J^* matrix is the optimal initial states and the minimum cycle cost. In order to determine the optimal state trajectories the model must be run forward in time using the optimal U^* controls to control the transmission. This is most easily accomplished by constructing a lookup table for the U^* matrices where the current states and time are inputs, and the controls are outputs. This forward-facing process further improves accuracy by continually interpolating between the discrete points optimized in the backwards stepping algorithm. When the DP algorithm is followed as specified, the results are guaranteed to be globally optimal down to the level the system is discretized. A sufficient level of discretization should be used such

that the DP results nearly converge with the true globally optimal solution. Determining the appropriate levels of discretization comes from experience. One simple (though computationally expensive) method of determining convergence is to increase the level of discretization and see how close the two results are.

A flowchart of the DP algorithm is shown in Figure 12.

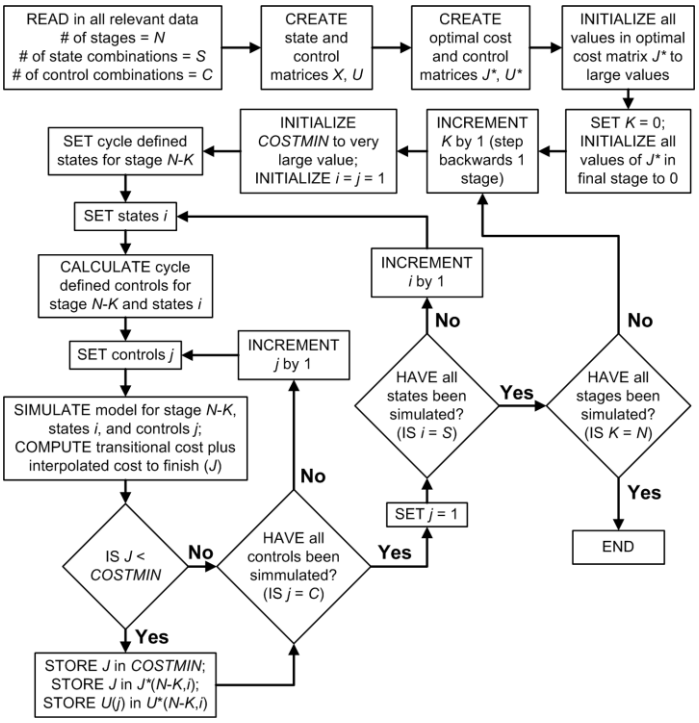

Figure 12: Dynamic programming flowchart

The results of this final run are shown in Figure 13 for the series hydraulic hybrid (Sprengel and Ivantysynova, 2014). An example of how the DP algorithm may use *a-prior* cycle knowledge can be seen by observing that in certain instances the optimal throttle input (generated by the engine speed controller) is relatively high, even though the vehicle is stopped. Closer inspection shows that the DP controller has determined that overall fuel consumption is minimized by storing energy in the accumulator while the vehicle is stopped in preparation for the acceleration phase. This is an example of the types of control strategies which a designer may not include in the dynamic model's controller during early development, but can have an impact on the system's performance.

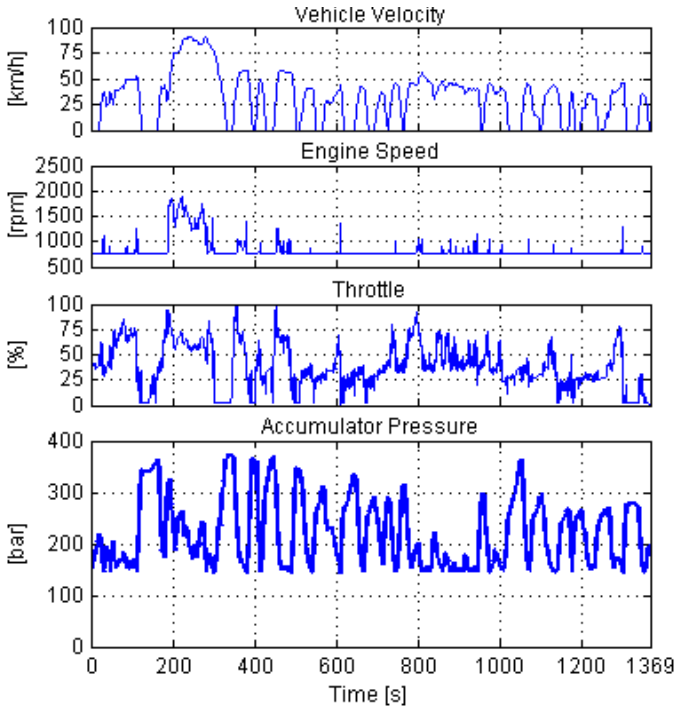

Figure 13: Globally optimal state trajectories

An engine operation map for the series hybrid is shown in Figure 14. This map includes the engine's Brake Specific Fuel Consumption (BSFC) with lower numbers indicating more efficient engine operation.

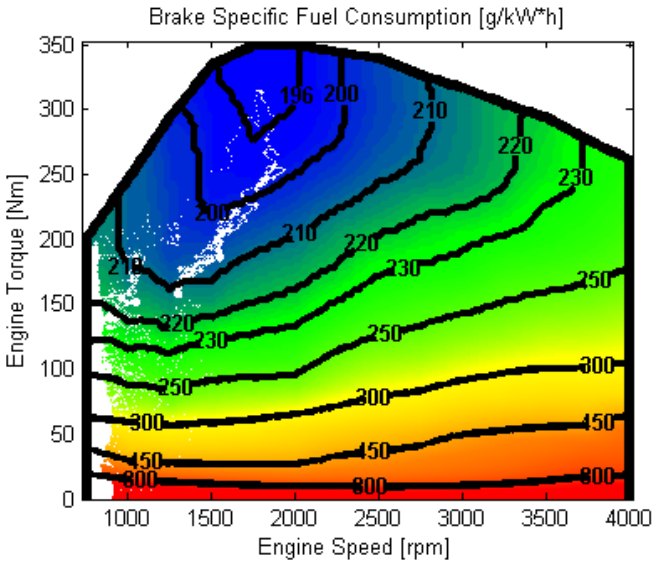

Figure 14: Globally optimal engine operation

Superimposed on the BSFC map is a histogram of the series hybrid engine's operation over the UDDS cycle, shown as white dots. These white dots indicate where the engine operated with their size proportional to the

cumulative duration of operation. This plot shows the series hybrid preferred to maintain a minimum engine speed except when operating near the engine's region of peak efficiency.

Implementing Dynamic Programming in the MATLAB Simulink Environment

Up until this point discussion of the DP algorithm has been kept platform agnostic. However more insight can be gained by now reviewing a specific implementation of DP. MathWork's MATLAB Simulink environment is one of the leading software packages for modeling and evaluating dynamic systems. In many cases the evaluation of novel system concepts begins with a Simulink model. A key advantage of the DP approach described in this paper is that it can be applied directly to these existing MATLAB Simulink based models with only a few minor modifications. This not only minimizes the time/effort required to apply DP, but also eliminates the need to greatly simplify the model (i.e. reduce fidelity) which may be necessary with matrix based state space formulations often used with other DP approaches.

One specific implementation of DP developed by Czero personnel for the MATLAB Simulink environment contains three key components: a primary DP algorithm written as a MATLAB script, a separate MATLAB initialization file containing all the DP parameters for a specific case (states, controls, discretizations, time steps, solver settings, etc.), and a Simulink model containing the dynamic system model configured in a such a way as to minimize computational burden.

When developing this DP implementation specific emphasis was placed not only on flexibility/ease of application, but importantly also on computational efficiency. As previously noted the computational expense of DP limits its usefulness for certain applications. The earlier example of a series hydraulic hybrid had 2116 states $(\omega_{\text{eng}}: 46\ x\ P_{\text{acm}}: 46)$, for each of which 966 controls $(\omega_{\text{eng}_{\text{des}}}: 46\ x\ \beta_1: 21)$ were evaluated. This required ~2.044 million dynamic simulations per time step, for which there were 1369, resulting in a total of ~2.798 billion dynamic simulations. Clearly any technique which speeds this process is of great interest. Fortunately, DP is highly parallelizable with every state and control evaluation within a given stage completely independent of one another.

Parallelization, i.e. the process of performing tasks simultaneously rather than sequentially, can be used to significantly reduce overall DP runtime (wall time, though not total CPU time). Parallelization of the DP algorithm is accomplished by employing several techniques to simultaneously evaluate all state/control combinations within a given stage. The process begins by configuring the Simulink model in such a way that the system's only required inputs are the desired state/control combination, the J^* cost to finish matrix, and the current time step within the cycle. The model is then simulated for the prescribed duration and outputs a single value containing the total cost to finish the cycle from that state/control combination. In this way a single set of inputs results a single output value summarizing the optimal path (i.e. the embedding principle). From the primary DP algorithm's perspective, the Simulink model simply converts the inputs provided to it (state/control combination) into an output value (cost to finish). If the primary DP algorithm were to evaluate each of the input combinations sequentially, the process would be quite slow. However as each of the input/output combinations are completely independent of one another, parallelization techniques can be used to significantly improve computational performance.

As previously noted modeling the dynamic system in Simulink (as opposed to the matrix form of a traditional state space representation) has many advantages including increased fidelity and ease of implementation. Simulink, however, has the disadvantage of requiring substantial overhead (time) to initialize a model relative to the time required to simulate a single DP time step. This excessive overhead means that using Simulink to evaluate each state/control combination with an individual simulation (which is how Simulink is normally used) is computationally infeasible. This limitation can be addressed by placing the entire simulation model within a repeating subsystem block in Simulink (as proposed by Liu and Peng (2006)). The repeating subsystem block effectively duplicates the single plant

model as many times as specified. This enables a single Simulink model to be opened once, and then simultaneously evaluate several hundred thousand individual simulations. Effectively the primary DP algorithm can supply an array of state/control combinations to the Simulink model which then returns an array of output values, a significantly more computationally efficiency approach than discrete simulations. Depending on model complexity, simulation rates of 30-50k+ simulations per second per core have been achieved using this method.

Runtime can further be improved by simultaneously running simulations on multiple processors. MATLAB's Parallel Computing Toolbox enables a single primary algorithm to split the full set of simulations required for a DP stage into subsets which are then distributed to multiple processors (e.g. 8 on a PC, 64 on a server). Once the simulations are complete the results are sent back to the primary algorithm where the optimal controls are determined and recorded. An overview of the parallelized DP algorithm is shown in Figure 15.

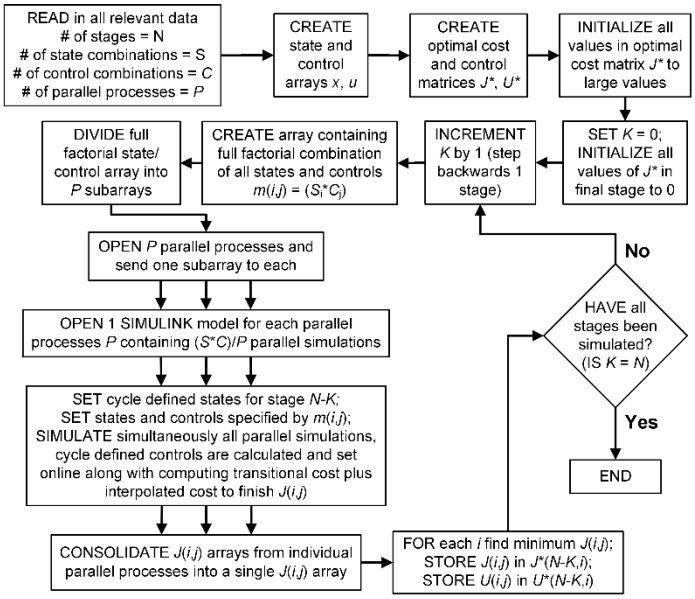

Figure 15: Parallelized dynamic programming flowchart

Conclusion

This works describes the application of discrete time dynamic programming to solve optimal state/control trajectory problems by applying the DP algorithm directly to dynamic simulations models. A key benefit of this approach is that it enables system designers to quickly and accurately assess the performance capabilities and characteristics of a potential architecture early in the design phase without necessarily having to understand how best to control it. By implementing DP early in the design phase system designers can:

- Ensure the superior system architecture and component sizes are selected
- Determine the performance capabilities of a system
- Discover more effective control strategies
- Minimize design iteration time and rework

How Czero Can Help You

Czero's focus is on helping innovative companies solve the toughest engineering problems through deep expertise, creative thinking, and sophisticated analysis tools. Among many other capabilities, Czero can apply the DP algorithm described in this white paper either directly to existing MATLAB Simulink based dynamic simulation models, or develop the models from first principles and existing libraries. This work can be performed as either an independent analysis task, or as part of a larger R&D project led by Czero.

Company Profile

Czero develops innovations for the automotive, defense, oil and gas, renewable energy, and clean technology industries.

Our award-winning engineers have 25+ years of experience working with innovation labs, startups, government agencies, and large OEMs in North America, Europe, Asia, and Australia.

Concept-to-prototype engineering R&D

Specializing in early-stage research and product development, Czero helps companies solve tough challenges and transform concepts into robust, tested prototypes of new technologies.

Services

- Mechanical design & solid modeling
- Dynamic modeling & simulation
- Finite element analysis (FEA) & computational fluid dynamics (CFD)

- Embedded controls
- Prototyping and testing
- Program & project management

R&D Specialties

- Advanced machine design
- Mechanical, electromechanical and electrohydraulic systems
- Energy conversion, efficiency and recovery
- High-bandwidth hydraulics
- Automotive powertrains
- Heavy-duty trucks
- Fuel systems
- Valve systems
- Hybrid vehicles

References

Bellman, **R**. 1956. Dynamic Programming and Lagrange Multipliers. Proceedings of the National Academy of Sciences of the United States of America, 42(10), 767.

Gong, **Q**., **Li**, **Y**. and **Peng**, **Z.R**., 2008. Trip-based optimal power management of plug-in hybrid electric vehicles. IEEE Transactions on vehicular technology, 57(6), pp.3393-3401.

Liu, **J**. and **Peng**, **H**. 2006. Control Optimization for a Power-Split Hybrid Vehicle. American Control Conference.

Sprengel, **M**. and **Ivantysynova**, **M**. 2014. Recent Developments in a Novel Blended Hydraulic Hybrid Transmission. SAE 2014 Commercial Vehicle Engineering Congress. Oct. 7-9, 2014. Rosemont, IL, USA. SAE Technical Paper 2014-01-2399.

Sprengel, **M**. 2015. Influence of Architecture Design on the Performance and Fuel Efficiency of Hydraulic Hybrid Transmissions. PhD thesis, Purdue University.

About the Author

Dr. Michael Sprengel is an R&D engineer and analyst at Czero in Fort Collins, Colorado. His focus is on the design, simulation, control, and optimization of novel and energy efficient systems. He has a BS in Mechanical Engineering from the Missouri University of Science and Technology, and both a MS in Mechanical Engineering, and a PhD, from Purdue University. While at Purdue he conducted his research at the Maha Fluid Power Research Center investigating novel and energy efficient hydraulic hybrid architectures and control strategies for on-road and off-highway applications.