1

Reduction of Storage Requirement by Checkpointing for Time-Dependent Optimal Control Problems in ODEs ⋆

Julia Sternberg[1] and Andreas Griewank[2]

[1] Technische Universität Dresden, Institut für Wissenschaftliches Rechnen, Germany, jstern@math.tu-dresden.de
[2] Humboldt-Universität zu Berlin, Institut für Mathematik, Germany, griewank@mathematik.hu-berlin.de

Summary. We consider a time-dependent optimal control problem, where the state evolution is described by an ODE. There is a variety of methods for the treatment of such problems. We prefer to view them as boundary value problems and apply to them the Riccati approach for non-linear BVPs with separated boundary conditions.

There are many relationships between multiple shooting techniques, the Riccati approach and the Pantoja method, which describes a computationally efficient stage-wise construction of the Newton direction for the discrete-time optimal control problem.

We present an efficient implementation of this approach. Furthermore, the well-known checkpointing approach is extended to a 'nested checkpointing' for multiple transversals. Some heuristics are introduced for an efficient construction of nested reversal schedules. We discuss their benefits and compare their results to the optimal schedules computed by exhaustive search techniques.

Key words: Optimal control, Newton method, Riccati approach, Nested checkpointing

1.1 Introduction

Consider the following unconstrained primal control problem

$$\min_{\mathbf{u}} \phi(\mathbf{x}(T)), \tag{1.1}$$

where the system is described by

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0. \tag{1.2}$$

⋆ Partially supported by the DFG Research Center MATHEON 'Mathematics for Key Technologies' in Berlin

Here, $\mathbf{x} : [0,T] \to \mathbf{R}^n$, $\mathbf{u} : [0,T] \to \mathbf{R}^m$, $f : \mathbf{R}^n \times \mathbf{R}^m \times [0,T] \to \mathbf{R}$, and $\phi : \mathbf{R}^n \to \mathbf{R}$.

The task is to find the function $\mathbf{u}(t)$ that minimizes (1.1). In order to characterize an optimal control function $\mathbf{u}(t)$ for the minimization problem (1.1) and (1.2) we consider the following adjoint state equation

$$\dot{\bar{\mathbf{x}}} = -H_{\mathbf{x}}^T = -f_{\mathbf{x}}^T\,\bar{\mathbf{x}}, \qquad \bar{\mathbf{x}}(T) = \phi_{\mathbf{x}}^T(\mathbf{x}(T)), \tag{1.3}$$

where the **Hamiltonian** function H is given by

$$H(\mathbf{x}(t), \mathbf{u}(t), \bar{\mathbf{x}}(t), t) = \bar{\mathbf{x}}^T(t)\, f(\mathbf{x}(t), \mathbf{u}(t), t). \tag{1.4}$$

Note, that here $\dot{\bar{\mathbf{x}}}$ represents the total time derivative of $\bar{\mathbf{x}}$ rather than a directional derivative as is customary in parts of the AD literature. At each point along the solution path the Hamiltonian function must be minimal with respect to the control value $\mathbf{u}(t)$. Therefore, we have the **First Order Necessary Optimality Condition**

$$\left(\frac{\partial H}{\partial u}\right)^T = 0, \qquad 0 \le t \le T, \tag{1.5}$$

for the optimal control problem (1.1) - (1.2).

A large variety of numerical methods for solving optimal control problems has been proposed and used in various fields of applications. Their relations amongst each other are often not very clear due to the lack of a generally accepted terminology. One fairly popular concept is to juxtapose approaches that first discretize and then optimize with those that first optimize and then discretize. Methods of the first type are sometimes called **direct** (see e.g. [1]) as they treat the discretized control problem immediately as a finite dimensional nonlinear program, which can be handed over to increasingly sophisticated and robust NLP codes. In the alternative approach one firstly derives optimality conditions in a suitable function space setting and then discretizes the resulting boundary value problem with algebraic side constraints. Often such **indirect** methods (see e.g. [7]) yield highly accurate results, but they have some disadvantages as well. Sometimes it is not possible to construct the boundary-value problem explicitly, as it requires that we can express the control function $\mathbf{u}$ in terms of $\mathbf{x}$ and $\bar{\mathbf{x}}$ from the relation (1.5). The second disadvantage is that often we have to find a very good initial guess including good estimates for the adjoint variables to achieve convergence to the solution. Alternatively one can solve the problem as a DAE with (1.5) representing a possibly discontinuous algebraic constraint.

Obviously, there is a range of intermediate strategies as one may for example discretize at first the controls and then the states only later. Christianson [3] makes a different distinction between direct and indirect methods, depending on whether the adjoint variables are integrated only backward or also forward. For stability reasons we consider here only the first option and show how the memory requirement can be kept within reasonable bounds nevertheless.

The BVP (1.2) - (1.3) is in general non-linear with separated (BC). We use a quasilinearization scheme to solve it iteratively. First, we linearize (1.2) - (1.3), and (1.5); after that we solve the resulting linear BVP using the Riccati approach.

This paper is organized as follows. Sect. 1.2 introduces the quasilinearization scheme. Nested checkpointing techniques and their main properties are discusses in Sect. 1.3. Sect. 1.4 gives a numerical example. Finally, in Sect. 1.5 we present some conclusions.

1.2 Quasilinearization techniques

In the present section we introduce the quasilinearization techniques, which can be applied for a stable solution of the optimal control problem (1.1) - (1.2).

1.2.1 Quasilinearization scheme

We linearize (1.2), (1.3), and (1.5) about a reference solution $\mathbf{x}(t)$, $\mathbf{u}(t)$, $\bar{\mathbf{x}}(t)$ and obtain the following equations for variations $\delta\mathbf{x}(t)$, $\delta\bar{\mathbf{x}}(t)$, and $\delta\mathbf{u}(t)$

$$\delta\dot{\mathbf{x}} - f_{\mathbf{x}}\delta\mathbf{x} - f_{\mathbf{u}}\delta\mathbf{u} = 0, \tag{1.6}$$

$$\delta\dot{\bar{\mathbf{x}}} + H^T_{\mathbf{xx}}\delta\mathbf{x} + H^T_{\mathbf{xu}}\delta\mathbf{u} + H^T_{\mathbf{x}\bar{\mathbf{x}}}\delta\bar{\mathbf{x}} = 0, \tag{1.7}$$

$$H^T_{\mathbf{u}} + H^T_{\mathbf{ux}}\,\delta\mathbf{x} + H^T_{\mathbf{uu}}\,\delta\mathbf{u} + H^T_{\mathbf{u}\bar{\mathbf{x}}}\,\delta\bar{\mathbf{x}} = 0, \tag{1.8}$$

with the linearized initial and terminal conditions

$$\begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}(0) \\ \delta\bar{\mathbf{x}}(0) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ -\phi_{\mathbf{xx}}(T) & I \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}(T) \\ \delta\bar{\mathbf{x}}(T) \end{pmatrix} = -\begin{pmatrix} \mathbf{x}(0) - \mathbf{x}_0 \\ \bar{\mathbf{x}}(T) - \phi^T_{\mathbf{x}}(T) \end{pmatrix}. \tag{1.9}$$

After expressing $\delta\mathbf{u}$ in terms of $\delta\mathbf{x}$ and $\delta\bar{\mathbf{x}}$ from the relation (1.8) we obtain

$$\delta\mathbf{u} = -H^{-1}_{\mathbf{uu}}\,\left(H^T_{\mathbf{u}} + H^T_{\mathbf{ux}}\,\delta\mathbf{x} + H^T_{\mathbf{u}\bar{\mathbf{x}}}\,\delta\bar{\mathbf{x}}\right). \tag{1.10}$$

Substitution of this expression into (1.6 - 1.7) yields the following linear BVP

$$\begin{pmatrix} \delta\dot{\mathbf{x}} \\ \delta\dot{\bar{\mathbf{x}}} \end{pmatrix} = S(t) \begin{pmatrix} \delta\mathbf{x} \\ \delta\bar{\mathbf{x}} \end{pmatrix} + q(t), \tag{1.11}$$

where

$$S(t) = \begin{pmatrix} S^{11} & S^{12} \\ S^{21} & S^{22} \end{pmatrix} = \begin{pmatrix} f_{\mathbf{x}} - f_{\mathbf{u}}\,H^{-1}_{\mathbf{uu}}\,H_{\mathbf{xu}} & \mid -f_{\mathbf{u}}\,H^{-1}_{\mathbf{uu}}\,f^T_{\mathbf{u}} \\ -H_{\mathbf{xx}} + H_{\mathbf{ux}}\,H^{-1}_{\mathbf{uu}}\,H_{\mathbf{xu}} & \mid H_{\mathbf{ux}}\,H^{-1}_{\mathbf{uu}}\,f^T_{\mathbf{u}} - f^T_{\mathbf{x}} \end{pmatrix} \tag{1.12}$$

is the system matrix, and

$$q(t) = \begin{pmatrix} q^1 \\ q^2 \end{pmatrix} = \begin{pmatrix} -f_{\mathbf{u}} \, H_{\mathbf{uu}}^{-1} \, H_{\mathbf{u}}^T \\ H_{\mathbf{ux}} \, H_{\mathbf{uu}}^{-1} \, H_{\mathbf{u}}^T \end{pmatrix} \tag{1.13}$$

is the non-homogeneous part. The BC for this problem are given by the relation (1.9). Rather than solving this linear BVP using collocation or another 'global' discretization scheme we prefer the Riccati approach which computes the solution in a sequence of forward and backward sweeps through the time interval $[0, T]$. In order to achieve a suitable decoupling of the solution components we consider a linear transformation of the form

$$\begin{pmatrix} \delta\mathbf{x}(t) \\ \delta\bar{\mathbf{x}}(t) \end{pmatrix} = \begin{pmatrix} I & 0 \\ K(t) & I \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}(t) \\ a(t) \end{pmatrix}. \tag{1.14}$$

In order to determine a suitable $K(t)$ and the corresponding $a(t)$ we substitute the $\begin{pmatrix} \delta\mathbf{x} \\ \delta\bar{\mathbf{x}} \end{pmatrix}$ in terms of $\begin{pmatrix} \delta\mathbf{x} \\ a \end{pmatrix}$ in (1.11), which yields

$$\frac{d}{dt} \begin{pmatrix} I & 0 \\ K & I \end{pmatrix} \begin{pmatrix} \delta\mathbf{x} \\ a \end{pmatrix} = \begin{pmatrix} S^{11} & S^{12} \\ S^{21} & S^{22} \end{pmatrix} \begin{pmatrix} I & 0 \\ K & I \end{pmatrix} \begin{pmatrix} \delta\mathbf{x} \\ a \end{pmatrix} + \begin{pmatrix} q^1 \\ q^2 \end{pmatrix}. \tag{1.15}$$

Now if we choose $K(t)$ as the solution of the Riccati equation

$$\dot{K}(t) = S^{21} - K(t)S^{11} + S^{22}K(t) - K(t)S^{12}K(t), \tag{1.16}$$

then the new variables $(\delta\mathbf{x}, a)$ satisfy the block system

$$\begin{pmatrix} \delta\dot{\mathbf{x}} \\ \dot{a} \end{pmatrix} = \left(\begin{array}{c|c} S^{11} + S^{12}K & S^{12} \\ 0 & S^{22} - KS^{12} \end{array} \right) \begin{pmatrix} \delta\mathbf{x} \\ a \end{pmatrix} + \begin{pmatrix} q^1 \\ q^2 - Kq^1 \end{pmatrix}, \tag{1.17}$$

with the separated BC

$$\begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}(0) \\ a(0) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ -\phi_{\mathbf{xx}}(T) + K(T) & I \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}(T) \\ a(T) \end{pmatrix} = - \begin{pmatrix} \mathbf{x}(0) - \mathbf{x}_0 \\ \bar{\mathbf{x}}(T) - \phi_{\mathbf{x}}^T(T) \end{pmatrix}. \tag{1.18}$$

This approach leads to the following conceptual algorithm.

Algorithm 1 Quasilinearization scheme for solving optimal control problems using the Riccati method
Choose initial control trajectory $\mathbf{u}^0(t)$, $t \in [0,T]$, $k = 0$.
Do:
  Original initialization:
    $\mathbf{x}^k(0) = \mathbf{x}_0$.
  Original sweep: $t : 0 \to T$
    Integrate forward $\dot{\mathbf{x}}^k = f(\mathbf{x}^k(t), \mathbf{u}^k(t), t)$.
  Adjoint initialization:
    Set $\bar{\mathbf{x}}^k(T) = \phi_{\mathbf{x}}^T(T)$.
    Set $K(T) = \phi_{\mathbf{xx}}(T)$ *and* $a(T) = 0$.
  Adjoint sweep: $t : T \to 0$
    Integrate backward $\dot{\bar{\mathbf{x}}}^k = -f_{\mathbf{x}}^T(\mathbf{x}^k(t), \mathbf{u}^k(t), t)\, \bar{\mathbf{x}}^k$.
    Integrate backward $\dot{K}(t) = S^{21} - K(t)S^{11} + S^{22}K(t) - K(t)S^{12}K(t)$.
    Integrate backward $\dot{a}(t) = \left(-K(t)S^{12} + S^{22}\right) a(t) - K(t)q^1 + q^2$.
  Final initialization:
    Set $\delta\mathbf{x}^k(0) = 0$.
  Final sweep: $t : 0 \to T$
    Integrate forward $\delta\dot{\mathbf{x}}^k = (S^{11} + S^{12}K)\delta\mathbf{x}^k + S^{12}a + q^1$.
    Evaluate $\delta\mathbf{u}^k = -H_{\mathbf{uu}}^{-1}\left(H_{\mathbf{u}}^T + H_{\mathbf{ux}}^T\,\delta\mathbf{x}^k + H_{\mathbf{u}\bar{\mathbf{x}}}^T(K\delta\mathbf{x}^k + a)\right)$.
    $\mathbf{u}^{k+1}(t) = \mathbf{u}^k(t) + \delta\mathbf{u}^k(t)$.
  $\mathbf{k} = \mathbf{k} + \mathbf{1}$.
While: $\|\delta\mathbf{u}^k(t)\|_2 \geq TOL$ *and* $k < MAX\_ITER$.

Discretizing the scheme in time one obtains Pantoja's method [2, 8], which represents a computationally efficient stage-wise construction of the Newton direction for the discrete-time optimal control problem. Moreover, this scheme can also be viewed as Newton's method applied to the solution of the nonlinear BVP (1.2), (1.3), and (1.5) using a particular LU-matrix factorization. The relation between Algorithm 1 and Pantoja's method is discussed in detail in [9].

1.2.2 Information flow by the Quasilinearization scheme

From Algorithm 1 we can see that each iteration of the quasilinearization scheme consists of three sweeps through the time window $[0, T]$, which are referred to as **original**, **adjoint** and **final** sweep. In Fig. 1.1 the dimensions of the data objects flowing between these three sweeps are shown.

The horizontal arrows represent informational flow between the three sweeps that are represented by slanted lines. Two cameras pointing at the original and adjoint sweeps represent the information which has to be stored if the current composite state is saved as a checkpoint. Here $B(t)$ is a $n \times m$ matrix path that must be communicated from the adjoint to the final sweep.

In any case the adjoint sweep requires much more computational effort than the original and the final sweeps because it involves matrix computations and factorizations. The final sweep proceeds forward in time and propagates

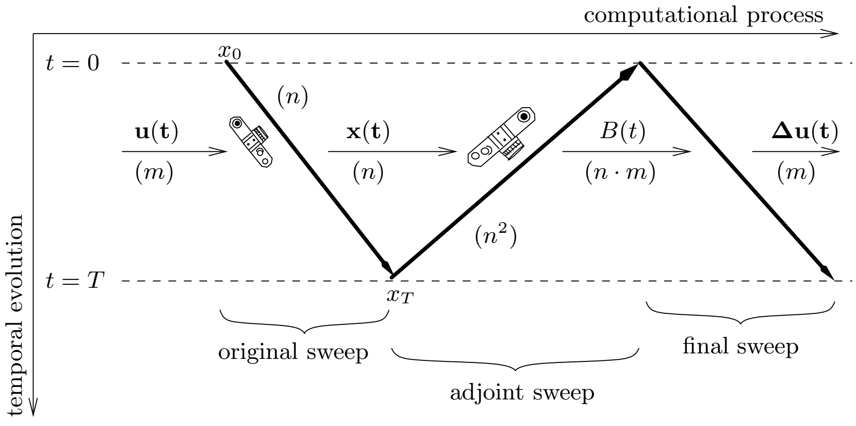

Fig. 1.1. Information flow for Riccati/Pantoja computation of Newton step

vectors of dimension $(n+m)$. Computations on the final sweep proceed as soon as required information from the previous sweeps is available. The final sweep can be combined with the original sweep of a subsequent Newton step.

The simplest strategy is to implement Algorithm 1 with straightforwardly storing all intermediate states of each sweep on a sequential data file and to restore them when they are needed. The memory requirement for the basic algorithm, where all intermediate values are stored, is of order $\mathcal{O}(l\,n^2)$, where l gives the number of time steps between 0 and T. However, this approach can be realized only when there is a sufficiently large amount of memory available. If this is not the case then we can apply checkpointing techniques.

As developed in [4, 5, 10] checkpointing means that not all intermediate states are saved but only a small subset of them is stored as checkpoints. In previous work we have treated cases where checkpoints are stored only for a reversal consisting of a single forward and an adjoint, or reverse sweep. But because of the triple sweep within each Newton iteration (see Algorithm 1 and Fig. 1.1) we are faced here with a new kind of checkpointing task. Since now checkpoints from various sweeps must be kept simultaneously, we refer to this situation as **nested checkpointing**.

Since the information to be stored on the original sweep differs from that needed on the adjoint sweep, we have two classes of checkpoints. Hence, we call the checkpoints **thin** on the original sweep and **fat** on the adjoint sweep. Thin checkpoints save a state space of dimension n, and fat checkpoints save a state space whose size has order n^2. While the length of steps may vary arbitrarily with respect to the physical time increment they represent, we assume throughout that the total number l of time steps is a priori known. When this is not the case, an upper bound on l may be used, which results of course in some loss of efficiency. Fully adaptive nested reversal schedules are under development.

1.3 Nested reversal schedules

In the present section we introduce a formal concept for nested checkpointing. Some heuristics are introduced for an efficient construction of nested reversal schedules. We discuss their benefits and compare their results to the optimal schedules computed by exhaustive search techniques.

1.3.1 Formalism

Let us consider a multiple sweep evolution $\mathcal{E}_{3(l)}$ containing three sweeps. Each sweep consists of l consecutive time steps. An example of such an evolution is shown in Fig. 1.2. Time steps are shown as horizontal arrows. Their directions denote the information flow. Nodes denote different intermediate states. Each sweep is characterized by a specified direction, i.e. direction of horizontal

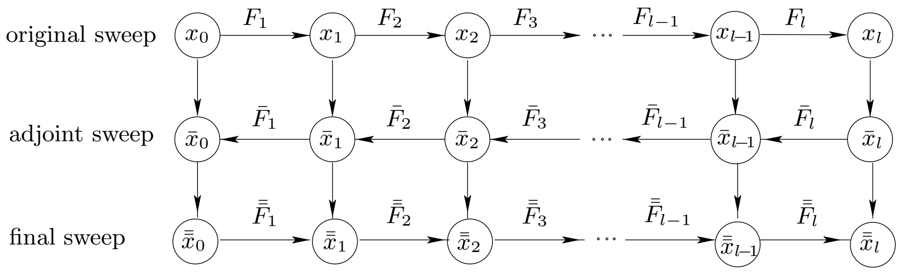

Fig. 1.2. Multiple sweep evolution $\mathcal{E}_{3(l)}$

arrows within a single sweep. A direction shows the corresponding information flow between neighboring intermediate states of a single sweep. Note that the information flow within a single sweep has a constant direction. An additional information flow exists between nodes, which are intermediate states of different successive sweeps. This is shown in Fig. 1.2 by vertical lines.

We denote intermediate states of the original, adjoint and final sweeps as x_i, $\bar{x}_i$, and $\bar{\bar{x}}_i$, $0 \leq i \leq l$, respectively. In the same manner we identify intermediate steps or time steps of various sweeps as F_i, $\bar{F}_i$, and $\bar{\bar{F}}_i$, $1 \leq i \leq l$. Then we have

$$x_i = F_i(x_{i-1}), \quad \bar{x}_{i-1} = \bar{F}_i(x_{i-1}, \bar{x}_i), \quad \bar{\bar{x}}_i = \bar{\bar{F}}_i(\bar{x}_i, \bar{\bar{x}}_{i-1}), \quad 1 \leq i \leq l. \tag{1.19}$$

We assume that dimensions of intermediate states within a single sweep are constant. Therefore, we denote them as

$$d \equiv \dim\{x_i\}, \quad \bar{d} \equiv \dim\{\bar{x}_i\}, \quad \bar{\bar{d}} \equiv \dim\{\bar{\bar{x}}_i\}, \quad 0 \leq i \leq l. \tag{1.20}$$

Moreover, we introduce evaluation costs, i.e. the computational effort for intermediate steps of different sweeps. We assume that within each single sweep we have uniform step costs, i.e. there exist three constants t, $\bar{t}$ and $\bar{\bar{t}}$, such that

$$t \equiv \mathrm{TIME}(F_i), \quad \bar{t} \equiv \mathrm{TIME}(\bar{F}_i), \quad \bar{\bar{t}} \equiv \mathrm{TIME}(\bar{\bar{F}}_i), \quad 1 \leq i \leq l. \tag{1.21}$$

Further, we assume that

$$\bar{d} >> d \qquad \text{and} \qquad \bar{t} >> t. \tag{1.22}$$

Thus, the dimension $\bar{d}$ of an intermediate state of an adjoint sweep is much higher than the dimension d of an intermediate state on the original sweep. Correspondingly the evaluation of time steps during an adjoint sweep is much more expensive than the evaluation of time steps during an original sweep. The assumptions (1.22) agree with the scenario presented in the Algorithm 1.

1.3.2 Definition of nested reversal schedules and its characteristics

The goal is to implement an evolution $\mathcal{E}_{3(l)}$ using nested checkpointing. The question is how to place different checkpoints to implement the evolution $\mathcal{E}_{3(l)}$ most efficiently. We call each possible strategy a **nested reversal schedule** because checkpoints are set and released at two different levels.

Thus, it is clearly not required to store intermediate states of the final sweep as checkpoints since information computed during this sweep is required just for subsequent time steps within this sweep, but not for previous sweeps. If only a restricted amount of memory is available, it is convenient to measure its size in terms of the number of fat checkpoints that it can accommodate. Since fat checkpoints, i.e. checkpoints of dimension $\bar{d}$, have to be stored during the adjoint sweep, we can use available memory on the original sweep to store thin checkpoints, i.e. checkpoints of dimension d, to reduce the total number of evaluated original steps F_i. On the adjoint sweep we remove thin checkpoints sequentially and store fat checkpoints instead of them as soon as required memory is available, i.e. as soon as a sufficient number of thin checkpoints is removed. We denote by $S(\mathbf{d_{3(l)}}, C)$ any admissible nested reversal schedule that can be applied to a multiple sweep evolution $\mathcal{E}_{3(l)}$ with a dimension distribution $\mathbf{d_{3(l)}} = (d, \bar{d}, \bar{\bar{d}})$ and a given number C of fat checkpoints. More formally we use the following definition.

Definition 1 (Nested Reversal Schedule $S(\mathbf{d_{3(l)}}, C)$). *Consider an evolution $\mathcal{E}_{3(l)}$ traversing l time steps in three alternative sweeps. Let $C \in \mathbb{N}$ fat checkpoints be available each of which can accommodate one intermediate state vector of the dimension $\bar{d}$, i.e. one intermediate state $\bar{x}_i$, $0 \leq i \leq l$. Moreover, assume that checkpoints can be stored during original and adjoint sweeps, provided sufficient memory is available. Assume that c thin checkpoints can be stored in place a single fat one, i.e. $\bar{d} = c\, d$. Then a* **nested reversal schedule** *$S(\mathbf{d_{3(l)}}, C)$ initializes $j = 0$ and $\bar{j} = l$, and subsequently performs a sequence of following basic actions*

$$\begin{aligned}
A\ &\equiv \textit{Increment } j \textit{ by } 1\\
\bar{A}\ &\equiv \textit{Decrement } \bar{j} \textit{ by } 1 \textit{ if } \bar{j}-j=1\\
W_i &\equiv \textit{Copy state } j \textit{ to a thin checkpoint } i \in \{0,1,\ldots,(C-1)c\}\\
\bar{W}_i &\equiv \textit{Copy state } \bar{j} \textit{ to a fat checkpoint } i \in \{1,2,\ldots,C\}\\
R_i &\equiv \textit{Reset state } j \textit{ to a thin checkpoint } i \in \{0,1,\ldots,(C-1)c\}\\
\bar{R}_i &\equiv \textit{Reset state } \bar{j} \textit{ to a fat checkpoint } i \in \{1,2,\ldots,C\}\\
D\ &\equiv \textit{Decrement } l \textit{ by } 1 \textit{ if } \bar{j}=1 \textit{ and } \bar{j}-j=1
\end{aligned}$$

until l *has been reduced to* 0.

It has to be arranged that each nested reversal schedule begins with the action R_0, such that the original state x_0 is read from the thin checkpoint 0.

One example of a nested reversal schedule $S(\mathbf{d_{3(9)}}, 2)$ for an evolution $\mathcal{E}_{3(9)}$ with the corresponding dimension distributions $\mathbf{d_{3(9)}} = (1,3,1)$ is shown in Fig. 1.3. Two fat checkpoints are available, i.e. 2 intermediate states on the adjoint sweep can be kept in memory simultaneously. Each of these states is of dimension $\bar{d} = 3$. Moreover, 3 thin checkpoints of dimension $d = 1$ can be stored instead of a single fat one. Further, the original state x^0 is stored as an additional 0th thin checkpoint.

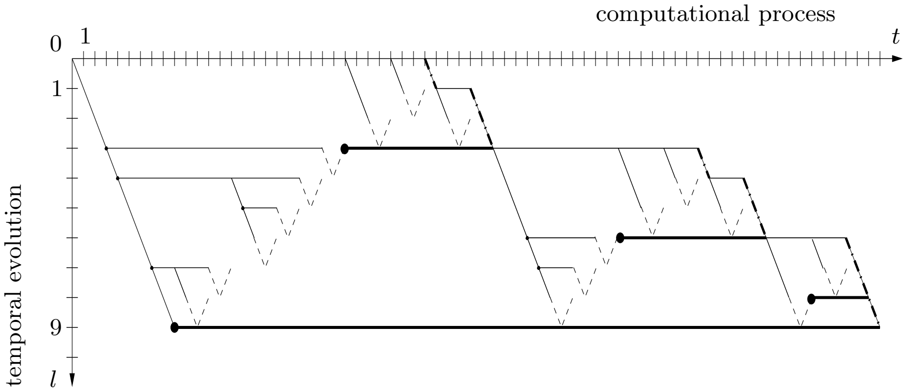

Fig. 1.3. Example of nested reversal schedule $S(\mathbf{d_{3(9)}}, 2)$

In Fig. 1.3 physical steps are plotted along the vertical axis, and time required for the implementation of the evolution $\mathcal{E}_{3(9)}$ measured in number of executed steps is represented by the horizontal axis. Hence, the horizontal axis can be thought of as a computational axis. Each solid thin horizontal line including the horizontal axis itself represents a thin checkpoint, i.e. a checkpoint of dimension $d = 1$. Each solid thick horizontal line represents a fat checkpoint, i.e. a checkpoint of dimension $\bar{d} = 3$. Solid slanted thin lines represent original steps F_i, whereas adjoint steps $\bar{F}_i$ are visualized by dotted slanted lines. Final steps $\bar{\bar{F}}_i$ are drawn by slanted dashed-dotted thick lines.

One starts with the action R_0 restoring the original state x_0 from the 0th thin checkpoint. Three actions A are executed by performing three original steps F_1, F_2, and F_3 consecutively. The state x_3 is stored into the first thin checkpoint by the action W_1. Now again the action A is applied to perform one original step F_4, and the state x_4 is stored into the second thin checkpoint by the action W_2. Further another three original steps are executed by the three actions A, and the state x_7 is stored in the third thin checkpoint. Then two further original steps are evaluated by the two actions A. Finally the state $\bar{x}_9$ is initialized and is stored in the first fat checkpoint by the action $\bar{W}_1$. The adjoint sweep is started by this action.

Further, the state x_7 is restored from the third thin checkpoint by the action R_3, the state x_8 is reevaluated by the action A, and the states $\bar{x}_8$ and $\bar{x}_7$ are evaluated by the application of the action $\bar{A}$ twice. In this manner we come to the state $\bar{x}_3$, which is stored in the second fat checkpoint. On the way backward all thin checkpoints are removed, all fat checkpoints are occupied consecutively, and we have no more memory available to store fat or even thin checkpoints. Then one goes back to the adjoint state $\bar{x}_1$ by reevaluating required intermediate states. Consequently the first final step $\bar{\bar{F}}_1$ is performed. Further, one stores the current original state x_1 in the 0th thin checkpoint and continues in the same manner to execute all other final steps $\bar{\bar{F}}_2, ..., \bar{\bar{F}}_9$.

Using the nested reversal schedule $S(\mathbf{d_{3(9)}}, 2)$ from Fig. 1.3 one needs to perform 28 original steps F_i, 18 adjoint steps $\dot{F}_i$, and 9 final steps $\bar{\bar{F}}_i$.

Definition 2 (Repetition Numbers). *Consider a nested reversal schedule $S(\mathbf{d_{3(l)}}, C)$. The* **repetition numbers** *$r_i \equiv r(i)$, $\bar{r}_i \equiv \bar{r}(i)$, and $\bar{\bar{r}}_i \equiv \bar{\bar{r}}(i)$, defined as functions*

$$r, \bar{r}, \bar{\bar{r}} : [1, l] \to \mathbb{N}, \tag{1.23}$$

count how often the ith original, the ith adjoint and the ith final step, is evaluated during the execution of the nested reversal schedule $S(\mathbf{d_{3(l)}}, C)$.

Provided a schedule is admissible in the sense that given $\mathbf{d_{3(l)}}$, C, and the initial l, it successfully reduces l to 0, its total runtime complexity can be computed from the additional problem parameters $\mathbf{t_{3(l)}} = (t, \bar{t}, \bar{\bar{t}})$. The temporal complexity of a nested reversal schedule $S(\mathbf{d_{3(l)}}, C)$, i.e. the run-time effort required to execute this nested reversal schedule can be computed as

$$\mathbf{T}(S(\mathbf{d_{3(l)}}, C), \mathbf{t_{3(l)}}) = t \sum_{i=1}^{l} r_i + \bar{t} \sum_{i=1}^{l} \bar{r}_i + \bar{\bar{t}} \sum_{i=1}^{l} \bar{\bar{r}}_i. \tag{1.24}$$

The optimal nested reversal schedule from the set of all admissible nested reversal schedules is required to minimize the evaluation cost, i.e. to achieve

$$\begin{aligned}&\mathbf{T}_{min}(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C) \equiv \\ &\min\left\{\mathbf{T}(S(\mathbf{d_{3(l)}}, C), \mathbf{t_{3(l)}}),\ S(\mathbf{d_{3(l)}}, C) \text{ is admissible}\right\}.\end{aligned} \tag{1.25}$$

The set of optimal nested reversal schedules is denoted by $S_{min}(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$, so that

$$\mathbf{T}(S_{min}(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)) \equiv \mathbf{T}_{min}(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C). \tag{1.26}$$

Now we face the task of constructing an appropriate optimal nested reversal schedule $S_{min}(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$. By brute force an optimal nested reversal schedule can be constructed using an exhaustive search algorithm (for more details see [9]). Using this approach one examines all possible distributions for thin and fat checkpoints and chooses from them the most efficient one. Clearly, such an exhaustive search is very expensive. In contrast to the situations for simple reversals involving only an original and an adjoint sweep, we have not been able to find a closed form characterization of optimal reversal schedules. Therefore, we have developed a heuristic for the construction of appropriate nested reversal schedules.

1.3.3 Heuristic

The intent of this heuristic is to restrict slightly the placements of thin checkpoints. Due the assumption (1.22), it is more convenient to reduce the freedom of movement of thin checkpoints, since even a considerable increment of the number of evaluated original steps does not cause a significant increase in the resulted evaluation cost wrt. the minimal evaluation cost $\mathbf{T}_{min}(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$ (accordingly (1.22) and (1.24)). Since one fat checkpoint can be stored as soon as the required memory is available, we store thin checkpoints such that after the removal of a sufficient number of thin checkpoints (c thin checkpoints), a corresponding fat checkpoint has to be stored at the same moment. Therefore, a nested reversal schedule can be decomposed into two nested subschedules and one simple reversal schedule as shown in Fig. 1.4.

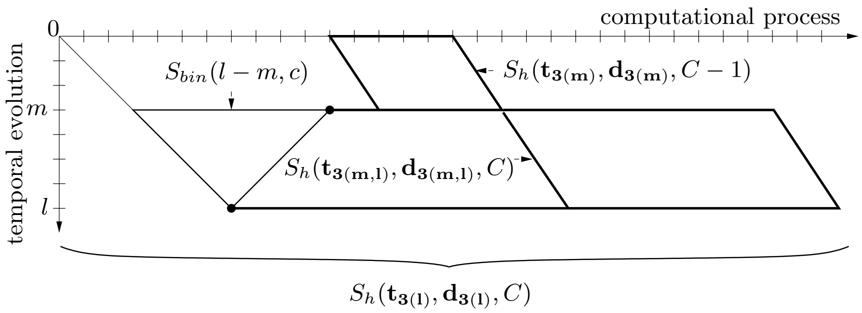

Fig. 1.4. Decomposition of a nested reversal schedule $S_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$

Here, $S_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$ denotes a corresponding nested reversal schedule, constructed using the heuristic described above. This schedule is decomposed into two parts $S_h(\mathbf{t_{3(m)}}, \mathbf{d_{3(m)}}, C-1)$ and $S_h(\mathbf{t_{3(m,l)}}, \mathbf{d_{3(m,l)}}, C)$ as shown in

Fig. 1.4 by the storing of the second fat checkpoint. An adjoint state stored in the second fat checkpoint corresponds to m. $S_{bin}(l-m, c)$ denotes the binomial reversal schedule with up to c checkpoints, which is applied for the reversal of $(l-m)$ original steps using minimal run-time and memory requirement (for details see e.g. [9]).

Then, an appropriate nested reversal schedule $S_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$ can be constructed recursively by minimizing the evaluation cost

$$T_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C) = \min_m \big\{T_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C, m)\big\} = \min_m \big\{T_h(\mathbf{t_{3(m)}}, \mathbf{d_{3(m)}}, C-1) \\ + T_h(\mathbf{t_{3(m,l)}}, \mathbf{d_{3(m,l)}}, C) + t_{add}(\mathbf{t_{3(m,l)}}, \mathbf{d_{3(m,l)}}, c)\big\}, \tag{1.27}$$

where $t_{add}(\mathbf{t_{3(m,l)}}, \mathbf{d_{3(m,l)}}, c)$ denotes an additional run-time effort required for placing $\bar{\mathbf{x}}_m$ in the second fat checkpoint. From (1.27) it is clear that the evaluation cost $T_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$ and consequently a corresponding nested reversal schedule $S_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$ can be evaluated using dynamic programming.

Instead of using dynamic programming we have developed a **Local-Descent Method**, which allows us to construct $S_h(\mathbf{t_{3(l)}}, \mathbf{d_{3(l)}}, C)$ using a linear run-time and memory requirement with respect to a number l of time steps and a number C of fat checkpoints (for details see [9]).

1.4 Numerical Example

We consider a control problem that describes the laser surface hardening of steel (see [6]). The mode of operation of this process is depicted in Fig. 1.5. A laser beam moves along the surface of a workpiece, creating a heated zone

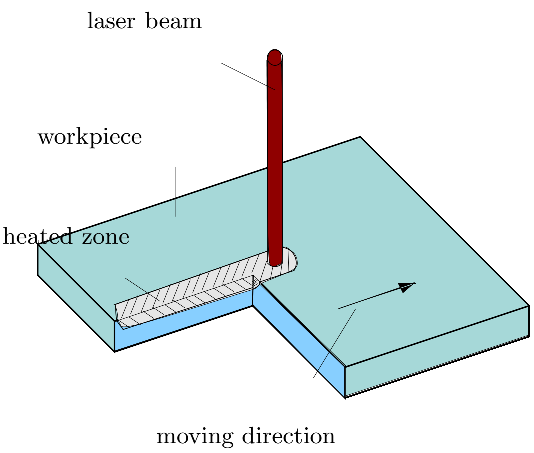

Fig. 1.5. Sketch of a laser hardening process

around its trace. The heating process is accompanied by a phase transition, in which the high temperature phase in steel, called austenite, is produced. Since one usually try to keep the moving velocity of the laser beam constant, the most important control parameter is the laser energy. Whenever the temperature in the heated zone exceeds the melting temperature of steel, the work-piece quality is destroyed. Therefore, the goal of surface hardening is to achieve a desired hardening zone, in our case described by a desired phase distribution a_d of austenite inside the workpiece Ω, but to avoid a melting of the surface. Hence we consider an optimal control problem with the cost functional $J(u)$ defined as

$$J(u) = \frac{\beta_1}{2} \int_\Omega (a(x,T) - a_d(x))^2 dx + \frac{\beta_2}{2} \int_0^T \int_\Omega [\theta - \theta_m]_+^2 dx\, dt + \frac{\beta_3}{2} \int_0^T u^2 dt, \tag{1.28}$$

where u is the laser energy and β_i $i = 1, 2, 3$ are positive constants. The second term in (1.28) penalizes temperatures above the melting temperature θ_m.

Let $\Omega := [0, 5] \times [-1, 0]$ with Lipschitz boundary $Q = \Omega \times (0, T)$, $\Sigma = \partial\Omega \times (0, T), T = 5.25$. The system of state equations (1.29) - (1.33) consists of a semi-linear heat equation coupled with the initial-value problem for the phase transitions. a is the volume fraction of austenite, θ the temperature, τ a time constant and $[x]_+ = \max\{x, 0\}$ the positive part function. The equilibrium volume fraction a_{eq} is such that the austenite volume fraction increases during heating until it reaches some value $a < 1$. During cooling we have $a_t = 0$, and the value a is kept. The homogeneous Neumann conditions were assumed on the boundary. The term $-\rho L a_t$ describes the consumption of latent heat due to the phase transition. The term $u(t)\alpha(x, t)$ is the volumetric heat source due to laser radiation, where the laser energy $u(t)$ will serve as a control parameter. The density ρ, the heat capacity c_p, the heat conductivity k, and the latent heat L are assumed to be positive constants.

$$a_t = \frac{1}{\tau(\theta)}[a_{eq}(\theta) - a]_+, \qquad in\ Q, \tag{1.29}$$

$$a(0) = 0, \qquad in\ \Omega, \tag{1.30}$$

$$\rho c_\rho \theta_t - k\triangle\theta = -\rho L a_t + u\alpha, \qquad in\ Q, \tag{1.31}$$

$$\frac{\partial\theta}{\partial\nu} = 0, \qquad on\ \Sigma, \tag{1.32}$$

$$\theta(0) = \theta_0, \qquad in\ \Omega. \tag{1.33}$$

We study the following state and control constrained optimal control problem for the cost functional $J(u)$ as defined in (1.28):

$$\min J(u),\ s.t.\ (\theta, a, u)\ solves\ (1.29) - (1.33)\ and\ u \in U_{ad}, \tag{1.34}$$

where $U_{ad} = \{u \in L^2(0, T) : \|u\|_{L^2(0,T)} \le 2800\}$ is the closed, bounded, and convex set of admissible controls. The numerical implementation is obtained by a semi-implicit FE Galerkin scheme. The FE triangulation of Ω is done by a nonuniform mesh. The optimal control problem (1.34) is solved using the Quasilinearization scheme from the Sect. 1.2 (see Algorithm 1). Nested reversal schedules are utilized for the reduction of memory requirement during the implementation of the Algorithm 1 applied to the optimal control problem (1.34).

1.5 Conclusion and Outlook

The iterative solution of optimal control problems in ODEs by various methods leads to a succession of triple sweeps through the discretized time interval. The second (adjoint) sweep relies on information from the first (original) sweep, and the third (final) sweep depends on both of them. This flow of information is depicted in Fig. 1.1. Typically the steps on the adjoint sweep involve more operations and require more storage than the other two. In order to avoid storing full traces of the original and adjoint sweeps we consider nested reversal schedules that require only the storage of selected original and adjoint intermediate states called thin and fat checkpoints. The schedules are designed to minimize the overall execution time given a certain total amount of storage for the checkpoints. While we have not found a closed form solution for this discrete optimization problem we have developed a cheap heuristic for constructing nested reversals that are quite close to optimality. Here we demonstrated that the dependence on l can be arranged polylogarithmically [4] by nested checkpoint strategies. Consequently, the operations count also grows as a second power of $\log_C l$, which needs not result in an increase of the actual run-time due to memory effects.

We are currently applying the proposed scheduling schemes to laser hardening of steel [6] and other practical optimal control problems. As has been done in case of simple reversal schedules that involve only an original and an adjoint sweep our results should be extended to scenarios with nonuniform step costs and parallel computing systems.

References

1. C. Büskens and H. Maurer. Sensitivity analysis and real-time control of parametric optimal control problems using nonlinear programming methods. *Online Optimization of Large Scale Systems, Springer-Verlag*, pages 57–68, 2001.
2. B. Christianson. Cheap newton steps for optimal control problems: Automatic differentiation and pantoja algorithm. *Optimization Methods and Software*, 10:729–743, 1999.
3. B. Christianson. A self-stabilizing pantoja-like indirect algorithm for optimal control. *Optimization Methods and Software*, 16:131–149, 2001.
4. A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

5. A. Griewank. *Evaluating derivatives. Principles and techniques of algorithmic differentiation*. SIAM, Philadelphia, 2000.
6. D. Hömberg and S. Volkwein. Control of laser surface hardering by a reduced-order approach using proper orthogonal decomposition. *Mathematical and Computer Modelling*, 38:1003–1028, 2003.
7. H. Maurer and D. Augustin. Sensitivity analysis and real-time control of parametric optimal control problems using boundary value methods. *Online Optimization of Large Scale Systems, Springer-Verlag*, pages 17–56, 2001.
8. J. Pantoja. Differential dynamic programming and newton's method. *International Journal on Control*, 47:1539–1553, 1988.
9. J. Sternberg. Reduction of storage requirement by checkpointing for time-dependent optimal control problems. *Doktorarbeit, in preparation*.
10. J. Sternberg. Adaptive umkehrschemata für schrittfolgen mit nicht-uniformen kosten. *Diplomarbeit*, Techn. Univ. Dresden, 2002.