JiffyTune: Circuit Optimization Using Time-Domain Sensitivities

by Andrew R. Conn[1], Paula K. Coulman[2], Ruud A. Haring[3], Gregory L. Morrill[4], Chandu Visweswariah[5] and Chai Wah Wu[6]

[1] Department of Mathematical Sciences,
IBM T. J. Watson Research Center, P.O.Box 218,
Yorktown Heights, NY 10598, USA
Email: arconn@watson.ibm.com

[2] IBM Microelectronics Division, Building 045, Department F77S,
11400 Burnet Road,
Austin TX 78758, USA
Email: coulman@austin.ibm.com

[3] Microsystems Department,
IBM T. J. Watson Research Center, P.O.Box 218,
Yorktown Heights, NY 10598, USA
Email: ruud@watson.ibm.com

[4] IBM Microelectronics Division,
Department 16S, Bldg. 862-2, 1000 River Road,
Essex Junction VT 05452, USA.
Email: s667403@btvlabvm.vnet.ibm.com

[5] Computer-Aided Design and Verification,
IBM T. J. Watson Research Center, P.O.Box 218,
Yorktown Heights, NY 10598, USA
Email: chandu@watson.ibm.com

[6] Department of Mathematical Sciences,
IBM T. J. Watson Research Center, P.O.Box 218,
Yorktown Heights, NY 10598, USA
Email: chaiwah@watson.ibm.com

Ruud A. Haring and Chandu Visweswariah are Senior Members of the IEEE.
Chai Wah Wu is a Member of the IEEE.

JiffyTune: Circuit Optimization Using Time-Domain Sensitivities

Andrew R. Conn Paula K. Coulman Ruud A. Haring
Gregory L. Morrill Chandu Visweswariah Chai Wah Wu

Abstract

Automating the transistor and wire sizing process is an important step towards being able to rapidly design high-performance, custom circuits. This paper presents a circuit optimization tool that automates the tuning task by means of state-of-the-art nonlinear optimization. It makes use of a fast circuit simulator and a general-purpose nonlinear optimization package. It includes minimax and power optimization, simultaneous transistor and wire tuning, general choices of objective functions and constraints, and recovery from non-working circuits. In addition, the tool makes use of designer-friendly interfaces that automate the specification of the optimization task, the running of the optimizer and the back-annotation of the results of optimization onto the circuit schematic.

Particularly for large circuits, gradient computation is usually the bottleneck in the optimization procedure. In addition to traditional adjoint and direct methods we use a technique, called the adjoint Lagrangian method, which computes all the gradients necessary for one iteration of optimization in a single adjoint analysis.

This paper describes the algorithms, the environment in which they are used and presents extensive circuit optimization results. A circuit with 6,900 transistors, 4,128 tunable transistors, and 60 independent parameters was optimized in about 108 minutes of CPU time on an IBM Risc/System 6000, model 590.

Keywords. Circuit tuning, simulation, transistor sizing, nonlinear optimization, adjoints.

1 Introduction, motivation and previous work

Automating the circuit optimization process is an important step towards rapidly and robustly designing high-performance circuits. Particularly in the use of custom designs, manually sizing schematics for area, delay and power is an iterative, slow, tedious and error-prone approach with circuit simulation in the inner loop. The updating of transistor widths from one iteration to the next in this context relies on human intuition. However, the quest for optimal performance and the importance of short time-to-market make automatic circuit tuning increasingly important. In addition, automatic tuning has the benefit of facilitating design adaptation and re-use. Hence an automatic tuning (and retuning) capability is becoming crucial to the productive design of custom circuits.

In the case of gradient-based dynamic tuning, function and gradient values are determined by means of a dynamic (time-domain) analysis of the underlying circuit. In the case of static optimization, a static timing analysis is relied upon to analyze new iterates produced during the optimization process. If circuit blocks are modeled by analytic delay equations, these equations can be differentiated symbolically to determine gradients. Unfortunately, this procedure is not applicable to custom designs because of the lack of availability of delay models for arbitrary transistor-level circuits.

In any type of circuit tuning, the computation of gradients is often the bottleneck in the optimization procedure. Gradients are generally computed by the direct method [1] or the adjoint method [2]. The direct method can provide the gradients of all of the measurements with respect to a single parameter and thus requires as many simulations of the associated sensitivity circuit as the number of tunable parameters. By contrast, in the adjoint method, the gradients of one measurement with respect to all parameters can be obtained in a single simulation of the adjoint circuit but the simulation must be repeated as many times as the number of measurements.

There have been many attempts to automate the transistor sizing problem. One of the earliest papers to apply gradient-based optimization, albeit to a specific network, was that of Hachtel and Rohrer [3]. A later paper [4] suggested a more general approach that also was intended for gradient-based nonlinear optimization. The latter used adjoint gradient computations and exploited sparse matrix methods. One class of methods [5, 6] is based on static timing analysis [7]. The delay of each cell is available either as a pre-characterized analytic function of the transistor sizes or as an Elmore delay approximation. In particular, if the Elmore delay model [8, 9] is used, this overall delay is seen to be a posynomial function (a particular algebraic form, see [10]) of the transistor widths and geometric programming techniques apply. By a simple mapping of variables, the objective is converted to a convex function [5] and hence any minimum of the latter is guaranteed to be a global minimum.

The advantages of static-timing-based methods include efficiency, the ability to handle large designs and the fact that they do not require input patterns to carry out the tuning. One of the significant disadvantages with these methods is that they are not applicable to full-custom circuit designs, since static timing analyzers usually rely on pre-characterized library cells. In addition, the accuracy of static timing analysis is limited (in our experience, the accuracy is at best about $\pm 25\%$ when Elmore delays are used) making it unsuitable as a basis for tuning high-performance custom circuits. Finally, static timing analysis suffers from the false path problem, so the optimizer may be working hard to tune paths that are either irrelevant or can never be sensitized. Recently, power optimization has been proposed in this general framework [11]. Power is measured by probabilistic methods [12] and then approximated by a posynomial function. Further, simultaneous tuning of drivers and interconnect has been proposed in [13] and [14].

Tuning based on dynamic simulation overcomes many of the above limitations of static tuning. The accuracy is as good as the simulator employed, false paths are not a problem and the method is applicable to any custom circuitry that the simulator can analyze. However, appropriate input patterns must be provided by the user. These methods (e.g., [15, 16]) typically run SPICE in the inner loop to optimize such circuit performance functions as gain, area, delay and phase margin. However, using SPICE iteratively is computationally expensive and significantly limits the size of the circuit that can be tuned.

From an overall design perspective, static and dynamic methods complement each other at different stages of the methodology, depending on the type of design, and there is an important place for automatic tuning in both. In this paper, we present a method for tuning custom MOS circuits that uses dynamic simulation and gradient-based optimization. Unlike previous methods salient features include minimax and power optimization, simultaneous transistor and wire tuning, recovery from non-working circuits that might be encountered during the tuning (Section 2.1), manufacturability modes, and general choices of functions and constraints. We emphasize that the ability to compute gradients efficiently is crucial to the success of this approach. A prototype implementation of our method, JiffyTune, optimizes circuits in about the CPU time of one SPICE analysis.

2 Overview of JiffyTune

This section provides an overview of the various high-level software components of JiffyTune and their interactions, as depicted in Figure 1. The JiffyTune 'engine' solves the following problem. We are given a circuit schematic, input signals, a set of circuit performance requirements, and a list of tunable transistors and wires with initial sizes. We wish to determine the optimal assignment of widths to tunable transistors and wires in order to achieve the requirements. The user interface makes it convenient for the user to specify the problem, review and accept the results of optimization. The interface is based on the schematic representation of the problem to be tuned and enables the designer to use the full functionality of the tuning tool by essentially 'pointing and clicking.' Once the problem is defined, control is passed to the JiffyTune engine, where the administration is handled by the Jiffytune block. The tuning process is iterative. At each iteration the fast circuit simulator SPECS[17, 18] provides the current function values, and on demand, the corresponding sensitivities to LANCELOT[19, 20, 21]. LANCELOT is a large-scale, general-purpose nonlinear optimization package. Subsequently the optimizer attempts to determine a better point, or decides that the tuning is completed. In the former case we return to SPECS, while in the latter case we leave the inner block and return to the interface passing control back to the designer, who can choose to back-annotate the results onto the schematic. The use of a state-of-the-art optimization technique based on gradients, along with efficient simulation, gives the flexibility and power to tune larger circuits than have previously been tuned dynamically, in addition to the ability to obtain close to optimal circuits rapidly. Subsequent sections contain detailed descriptions of the individual components.

2.1 JiffyTune

The JiffyTune block in Figure 1 performs the administrative portion of the tuning task. The specification of circuit optimization problems is handled via a control file and a corresponding control file grammar. The control file contains the following information.

Parameters: This section contains a list of tunable transistors and wires, their initial widths and simple bounds. Tunable parameters can be ratioed to (i.e., declared to be a fixed multiple of) other tunable parameters and the user interface allows grouping of instances of similar structures so that they track each other during tuning. Thus, for example, the cells of an n-bit wide multiplexer can be grouped to ensure

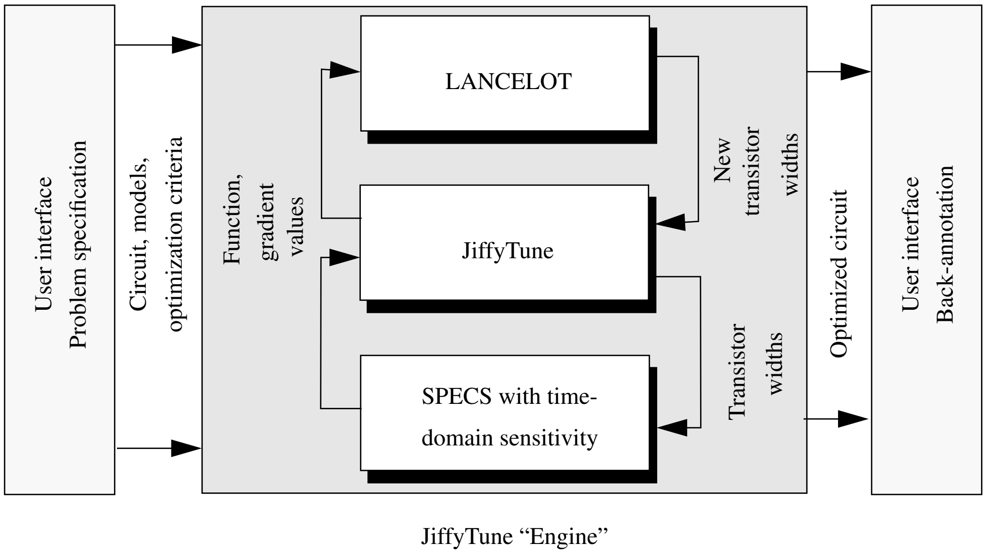


Figure 1: High-level view of JiffyTune.

that the cells stay identical through the tuning process, consequently lending themselves to a structured, regular layout.

Measurements and functions: A measurement is either a crossing-time, power or area. In the absence of layout information, area is modeled by the sum of the tunable parameter widths. Functions consist of any linear combination of measurements so, for example, delays and rise/fall times are typically the difference of two crossing times. Each function has a weight, a target and a relation. For example, a relation of 'less than' implies that this function should be less than the target value. Alternately, a function can be 'minimized' which means that the optimizer will treat it as an objective function. Weights can be used to explore various trade-offs in tuning the circuit; these are particularly important when functions of different quantities (area, delay, power) are being optimized. Any number of functions can be grouped into a minimax function. A minimax function implies that the largest of some number of functions needs to be minimized. For example, the statement of the problem might be to minimize the delay of the worst of three paths through a combinational logic block.

Controls: This section provides administrative information like the maximum number of iterations, the layout grid for rounding transistor widths at the end of optimization, and the location of the device model files.

JiffyTune reads the control file and internally represents the problem in a format that is understood by LANCELOT. JiffyTune also provides to LANCELOT a callable routine that will accept a set of wire and transistor widths, perform a SPECS simulation and return function and gradient values in the form required by LANCELOT. Then JiffyTune begins a LANCELOT optimization. At each iteration, JiffyTune keeps track of the best results so far. One of the main functions of the JiffyTune block is to chain rule and combine gradients to provide to LANCELOT the gradients of various functions with respect to independent variables only. Typically, 25 to 30 iterations are required for convergence. The default maximum number of iterations in JiffyTune is 50. Because the bulk of the cost is in the simulation (over 90% in our benchmark problem set), there is a high incentive for making heroic efforts to reduce the number of optimization iterations, even beyond what optimizers would typically do. For example, the recently implemented slack updating method described in Section 4.2 has led to fewer iterations being required in general. Further, to decrease the number of steps and the sensitivity to noisy simulation, LANCELOT is encouranged to take larger steps. This could lead to a non-working circuit (e.g., a circuit in which a measured signal transition doesn't even occur in the simulation) and a recovery mechanism is implemented whereby the step size is decreased and the simulation rerun.

2.2 SPECS

SPECS is a fast circuit simulator that uses simplified device models and event-driven techniques. JiffyTune calls SPECS in the inner loop to evaluate the circuit and provide function and gradient values. SPECS and its sensitivity computation capabilities are described in more detail in Section 3.

2.3 LANCELOT

LANCELOT[19, 20, 21] is a general-purpose large-scale nonlinear optimization package that handles simple bounds and general constraints. JiffyTune provides the problem description and initial transistor and/or wire sizes to LANCELOT. LANCELOT repeatedly calls SPECS with different parameter size settings, and builds a model of the 'performance surface' of the circuit. Details regarding LANCELOT and its application to the circuit tuning problem are provided in Section 4. In addition to LANCELOT, the optimization package MINOS [22] has been integrated into JiffyTune using the tools that accompany the optimization testing environment CUTE [23]. MINOS integration has been used only for comparisons and as a 'sanity check.'

3 SPECS and time-domain sensitivity computation

SPECS (Simulation Program for Electronic Circuits and Systems) is a fast circuit simulation program. SPECS is on the average 70x faster than AS/X, an IBM internal SPICE-like circuit analysis program [24]. SPECS achieves this speed by using simplified device models and event-driven techniques to efficiently simulate MOSFET circuits in the time-domain, and it has been used in production mode in numerous integrated circuit designs. The device modeling assumptions in SPECS restrict its relative timing accuracy to $\pm 5\%$, and hence JiffyTune can only tune to within this accuracy. Although JiffyTune uses SPECS to evaluate the circuit being optimized, this paper will not describe SPECS in any detail, except for the new adjoint Lagrangian computations. The reader is referred to [17], [18] and [25] for a more comprehensive explanation.

3.1 Sensitivity computation

SPECS uses simplified device models that consist of piecewise constant i-v characteristics in multiple dimensions and grounded, linear capacitances. These simplifications allow efficient, incremental time-domain sensitivity computation [26, 27, 28]. Both the adjoint method [2, 25] and the direct method [1] have been implemented. In the direct method, branch constitutive relations (device characteristics) are directly differentiated with respect to the sensitivity parameter of interest and the circuit reflecting these differentiated equations, called the sensitivity circuit, is solved to obtain the gradients. Since SPECS uses piecewise constant device models, the sensitivity circuit consists of disconnected capacitances for large sub-intervals of time, with occasional impulses of currents flowing between these capacitances at times corresponding to events in the nominal simulation. Thus the solution of the sensitivity circuit is extremely efficient. In the direct method, the sensitivities of all functions with respect to one parameter are computed with a single solution of the sensitivity circuit.

In the adjoint method, elements are replaced by adjoint equivalents based on Tellegen's theorem (see for example [2, 25]). Again, in the case of SPECS, the circuit is very simple and lends itself to efficient solutions. In this case, however, time is run backwards in the adjoint circuit, and the waveforms of the adjoint circuit are convolved with those of the nominal circuit to obtain the required sensitivities. The gradients of one function with respect to all parameters are computed in a single solution of the adjoint circuit. Hence, when there are sufficiently more parameters than functions to justify the overhead of convolution, the adjoint method is advantageous.

Note that once the approximation in the simplified device models is accepted, the computation of the functions and gradients is exact. After the sensitivity circuit is solved in either method, gradients are chain-ruled and combined to obtain the sensitivity of each function with respect to all ramifications of varying the tunable parameter. We remark that when the width of a transistor varies, its source and drain diffusion capacitance and all the intrinsic MOSFET parasitic capacitances change. Consequently, each of these is submitted as an internal sensitivity parameter and then all the gradients are postprocessed and combined appropriately. The flavor of these computations is captured by the following simplified equation.

$$\begin{aligned} \frac{df}{dW} &= \frac{\partial f}{\partial W_{eff}} \cdot \frac{dW_{eff}}{dW} + \frac{\partial f}{\partial CD_{total}} \cdot \frac{dCD_{total}}{dW} \\ &+ \frac{\partial f}{\partial CS_{total}} \cdot \frac{dCS_{total}}{dW} + \frac{\partial f}{\partial CG_{total}} \cdot \frac{dCG_{total}}{dW}, \end{aligned} \tag{1}$$

where f is the sensitivity function of interest, W is the transistor width (sensitivity parameter), W_{eff} is the effective width and CG_{total}, CS_{total} and CD_{total} are the sums of the capacitance components at the gate, source and drain nodes, respectively. Subsequently (1) is further expanded in terms of the device model parameters.

SPECS assumes that all tunable wires are modeled by RC circuits. To compute gradients with respect to wire parameters, all resistors and capacitors that depend on the parameter of interest are first identified. The gradients with respect to these element values are computed and then chain-ruled with the gradients of the element values with respect to the parameters of interest. Finally, the results of the chain ruling are summed over the RC elements of each wire model.

Voltage crossing sensitivities are computed as follows, see for example [29]. Let v_N be the node voltage that crosses the level v_{cross} at time t_{cross} in the nominal circuit. We want to find $\frac{dt_{cross}}{dp}$, where p is the sensitivity parameter. Hence

$$v_{cross} = v_N\big|_{t=t_{cross}}. \tag{2}$$

Differentiating,

$$\frac{dv_{cross}}{dp} = 0 = \frac{d}{dp} v_N(t,p) = \left.\frac{\partial v_N}{\partial t}\right|_{t=t_{cross}} \frac{dt_{cross}}{dp} + \left.\frac{\partial v_N}{\partial p}\right|_{t=t_{cross}}, \tag{3}$$

reflecting the fact that the node voltage is a function of time and the sensitivity parameter p. Thus

$$\frac{dt_{cross}}{dp} = -\frac{\left.\frac{\partial v_N}{\partial p}\right|_{t=t_{cross}}}{\left.\frac{\partial v_N}{\partial t}\right|_{t=t_{cross}}}. \tag{4}$$

Hence the required sensitivity is the negative of the quotient of the sensitivity of the node voltage to the parameter at the time of crossing and the slope of the node voltage in the nominal circuit at that same instant. This concept is referred to as an 'event functional' in [29].

3.2 Adjoint Lagrangians

The number of time-domain gradients computed during a typical JiffyTune run is often in the millions. Hence gradient computation must be extremely efficient to make this process feasible. To address this problem, particularly in the case of larger circuits, we introduce the concept of using adjoint Lagrangians for efficient gradient computation. As was implicit in [4] and [2], adjoints can be applied to any composite scalar functions. The novelty of the adjoint method proposed in this paper is that the merit function under consideration, the augmented Lagrangian function, is a nonlinear function of these measurements that also involves optimization parameters such as Lagrange multipliers and the penalty parameter. However, because of the close integration of the optimizer and the simulator, it is possible to excite the adjoint circuit in such a manner as to obtain the gradients of the merit function in a single adjoint simulation.

We first demonstrate the basic idea by means of a simple example. Referring to Figure 2, let us assume that the circuit to be optimized has just one input and one output. Consider the problem of minimizing d_1 subject to $d_2 = T$ where d_1 denotes the 50% crossing time of the falling transition at the single output of interest, d_2 is the 50% crossing time of the rising transition and T is a constant target value. The tunable parameters are $x_1, x_2, \cdots, x_n$. The output waveform is shown in Figure 2(a). Let us further assume that corresponding to this circuit the nonlinear optimizer builds an augmented Lagrangian merit function [30, 31] of the form

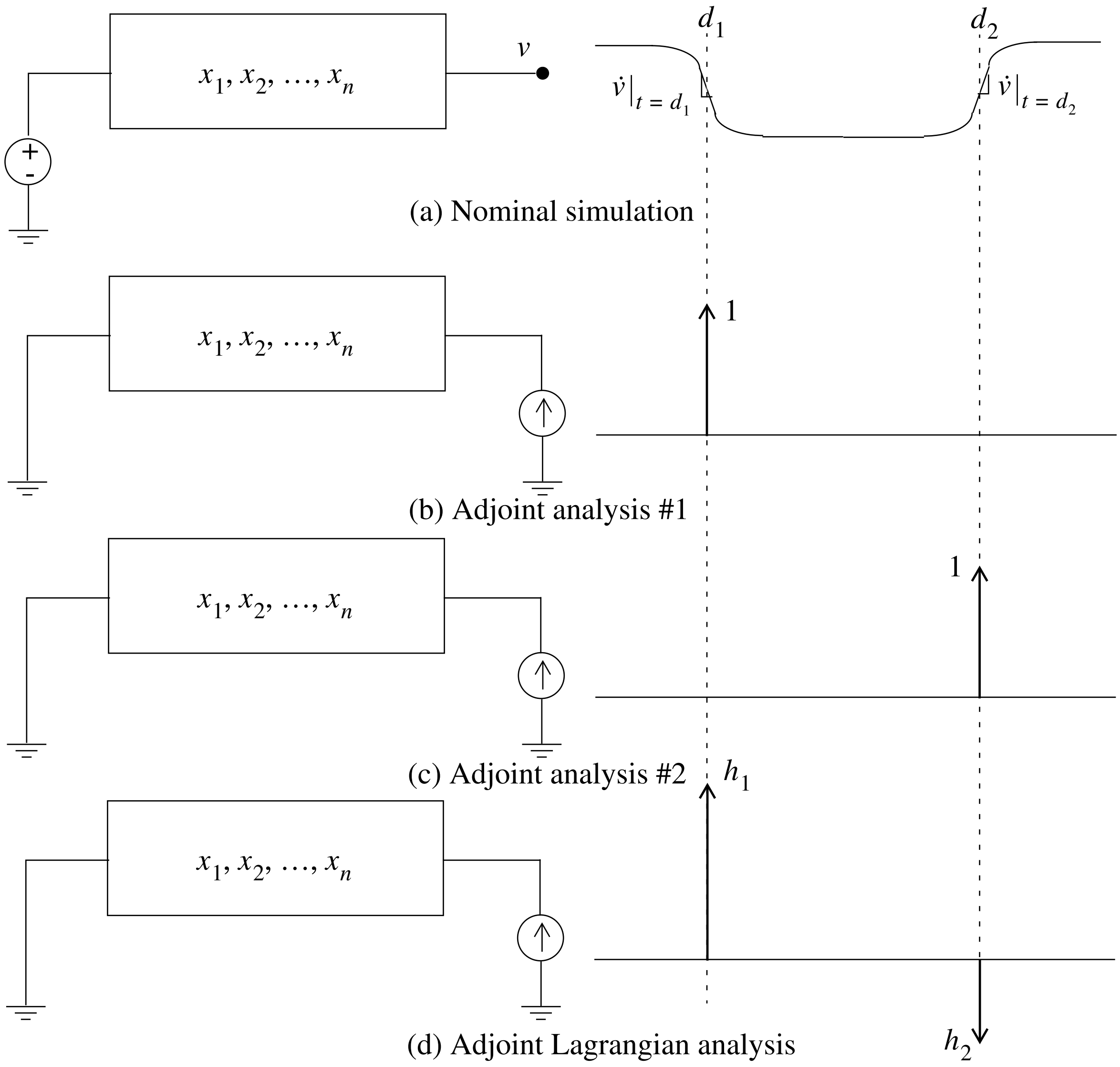


Figure 2: Demonstration of the adjoint Lagrangian formulation by means of an example.

$\Phi = d_1 + \lambda(d_2 - T) + \frac{1}{2\mu}(d_2 - T)^2$ where λ is the Lagrange multiplier corresponding to the constraint, and μ is a penalty parameter used to weight the quadratic augmentation of the Lagrangian. At each iteration, the simulator is required to compute d_1, d_2, $\partial d_1/\partial x_i$ and $\partial d_2/\partial x_i$ for all i. The quantities d_1 and d_2 are computed by a nominal transient analysis, as shown in Figure 2(a). We will first describe how a measurement-at-a-time adjoint analysis method is used to determine the sensitivities. Initially, an adjoint circuit is appropriately formed and configured as shown in Figure 2(b). The adjoint circuit is excited by a current source with a unit Dirac impulse at time corresponding to d_1, i.e., $\delta(t - d_1)$. Time and control are reversed during the analysis of the adjoint circuit and the nominal waveforms are convolved with the adjoint waveforms to yield

$$\left.\frac{\partial v}{\partial x_i}\right|_{t=d_1} = conv(x_i), \forall i, \tag{5}$$

where v is the output signal of interest and $conv(x_i)$ represents the convolution between the appropriate nominal and adjoint waveforms for each problem variable. The required gradients are then computed using the formula

$$\frac{\partial d_1}{\partial x_i} = -\left.\frac{\frac{\partial v}{\partial x_i}}{\frac{\partial v}{\partial t}}\right|_{t=d_1} = -\frac{\left.\frac{\partial v}{\partial x_i}\right|_{t=d_1}}{\dot{v}|_{t=d_1}} = \frac{-conv(x_i)}{\dot{v}|_{t=d_1}}, \tag{6}$$

where $\dot{v}|_{t=d_1}$ is the slope of the nominal voltage waveform at time d_1. Next, the adjoint analysis is repeated as shown in Figure 2(c) to similarly determine the gradients $\partial d_2/\partial x_i$ for all i. Finally, the gradients are assembled by the optimizer as follows

$$\frac{\partial \Phi}{\partial x_i} = \frac{\partial d_1}{\partial x_i} + \left[\lambda + \frac{(d_2 - T)}{\mu}\right] \frac{\partial d_2}{\partial x_i}. \tag{7}$$

The method described above requires two adjoint analyses and two sets of convolution integrals. Instead, our adjoint Lagrangian method recognizes that if one assumes that the coefficients of $\partial d_1/\partial x_i$ and $\partial d_2/\partial x_i$ are known, from (7), the gradients of the merit function can be considered as a linear combination of the gradients of the circuit measurements since one could rewrite it as

$$\frac{\partial \Phi}{\partial x_i} = \frac{\partial}{\partial x_i}\left[h_1 d_1 + h_2 d_2\right]. \tag{8}$$

However, although in general nonlinear, the coefficients h_1 and h_2 can be treated as constants once the nominal simulation has been completed. Thus two impulses of heights

$$h_1 = \frac{-1}{\dot{v}|_{t=d_1}} \tag{9}$$

and

$$h_2 = \frac{-1}{\dot{v}|_{t=d_2}} \left[\lambda + \frac{(d_2 - T)}{\mu} \right] \tag{10}$$

are applied during a single adjoint analysis. Note that the heights of the impulses are functions of the nominal simulation results $\left(\dot{v}|_{t=d_1}, \dot{v}|_{t=d_2} \text{ and } d_2 \right)$ and the parameters λ and μ from the optimization. The analysis and convolution integrals are carried out as before to obtain

$$\left. \frac{\partial \Phi}{\partial x_i} \right|_{t=d_1} = conv(x_i), \forall i. \tag{11}$$

Thus two adjoint analyses have been replaced by one by taking advantage of the fact that the optimizer needs only the gradient of Φ rather than the the gradients of the individual components of Φ. Hence, a single adjoint analysis is enough to determine the gradients of Φ with respect to all the variables of the problem, as shown in Figure 2(d).

In general, if an optimization problem involves m measurements, the adjoint Lagrangian method will obtain a speedup of $O(m)$ over a measurement-at-a-time adjoint analysis. Although the gradients of the composite merit function can be computed by this method, the gradients of the individual measurements are not computed and cannot be recovered and as we shall see below (in Sections 3.4 and 4.3) this limitation introduces some complications. The next section will describe in detail how the gradients of any differentiable scalar merit function of any number of circuit measurements can be computed with respect to any number of parameters by means of a single adjoint analysis.

3.3 Adjoint Lagrangian theory

This section will describe gradient computation by the adjoint Lagrangian method. The basic principle is as follows. The gradients of merit functions f_1 and f_2 can be computed by exciting the adjoint circuit with impulses p_1 and p_2 respectively. It turns out that the gradient of $f_1 + f_2$ can be computed by exciting the adjoint circuit with *both* impulses p_1 and p_2. By considering the merit function as a linear combination of several simple functions, we can calculate the gradient of the merit functions by exciting the adjoint circuit with several impulses together. This is more than just linear superposition as the impulses are applied to a *nonlinear* adjoint circuit.

The ensuing mathematical derivation will show that the gradients of an arbitrary differentiable merit function of several measurements can be computed by means of a single adjoint analysis. Scaled versions of the appropriate individual excitations for measurement-at-a-time adjoint analysis are simultaneously applied to the time-varying adjoint circuit, and gradients computed by means of convolution integrals. Suppose the merit function of interest is

$$\Phi = g(f_1, f_2, \cdots, f_F, c_1, c_2, \cdots, c_C), \tag{12}$$

where g is any differentiable function, the f_j are objective functions (one in the case of single-criterion optimization, several in the case of multi-criteria optimization) and the c_j are constraints. Further, let these objective functions and constraints be defined as differentiable functions

$$\begin{aligned} f_j &= f_j(m_1, m_2, \cdots, m_M),\ j &= 1, \ldots, F \\ c_j &= c_j(m_1, m_2, \cdots, m_M),\ j &= 1, \ldots, C \end{aligned} \tag{13}$$

of the circuit measurements m_k. We are required to find the gradient of Φ with respect to all the $x_i, i = 1, 2, \ldots, n$ parameters of the optimization. Now

$$\begin{aligned} \frac{\partial \Phi}{\partial x_i} &= \sum_{j=1}^{F} \frac{\partial g}{\partial f_j} \frac{\partial f_j}{\partial x_i} + \sum_{j=1}^{C} \frac{\partial g}{\partial c_j} \frac{\partial c_j}{\partial x_i}, \forall i \\ &= \sum_{j=1}^{F} \left(\frac{\partial g}{\partial f_j} \sum_{k=1}^{M} \frac{\partial f_j}{\partial m_k} \frac{\partial m_k}{\partial x_i} \right) + \sum_{j=1}^{C} \left(\frac{\partial g}{\partial c_j} \sum_{k=1}^{M} \frac{\partial c_j}{\partial m_k} \frac{\partial m_k}{\partial x_i} \right), \forall i. \end{aligned} \tag{14}$$

Rearranging the summations,

$$\frac{\partial \Phi}{\partial x_i} = \sum_{k=1}^{M} \frac{\partial m_k}{\partial x_i} \left[\sum_{j=1}^{F} \frac{\partial g}{\partial f_j} \frac{\partial f_j}{\partial m_k} + \sum_{j=1}^{C} \frac{\partial g}{\partial c_j} \frac{\partial c_j}{\partial m_k} \right], \forall i. \tag{15}$$

All the terms inside the square brackets of the right hand side of (15) are known once the nominal simulation has been completed, since the form of g, the forms of f_j and c_j, and the nominal values of m_k , the measurements, are known. Note that the terms within the square brackets can depend on the nominal simulation results, as well as optimization parameters such as slack variables, Lagrange multipliers, penalty parameters, etc.,. Now (15) can be rewritten as

$$\frac{\partial \Phi}{\partial x_i} = \sum_{k=1}^{M} h_k \frac{\partial m_k}{\partial x_i} = \frac{\partial}{\partial x_i} \left[\sum_{k=1}^{M} h_k m_k \right], \forall i, \tag{16}$$

since the h_k can be treated as constants after the nominal simulation. We have reduced the problem of finding the gradients of the merit function to that of finding the gradients of a linear combination of measurements, provided the corresponding coefficients are known, and the functions g, f_j and c_j are differentiable..

We will demonstrate how to pick adjoint circuit excitations so that the right hand side of (16) can be computed efficiently. Let each measurement be expressible as a time-domain convolution integral of the form

$$m_k = \int_{t_0}^{t_f} \{v_I, i_V\}\, p_k(\tau) d\tau, \tag{17}$$

where v_I are voltages of independent current sources, i_V are the currents of independent voltage sources, t_0 is the start time of the transient simulation, t_f the end time, $\{v_I, i_V\}$ denotes one of v_I and i_V and p_k is a time-domain function that will be used as the excitation in the adjoint circuit at the measurement point. Without loss of generality, all measurements can be written in terms of the voltages of independent current sources and currents of independent voltage sources, since a zero-valued current (voltage) source can always be added in parallel (series) with the voltage (current) to be measured. For example, a measurement which is a voltage value of node k at any time t is expressed as $m_k = \int_{t_0}^{t_f} v_{Ik}(\tau)\delta(\tau - t)d\tau$ so that p_k is a unit Dirac impulse at time corresponding to t. A measurement which is a crossing time (as in the example of Section 3.2) requires $p_k(\tau) = \frac{-\delta(\tau - t_{cross})}{\dot{v}|_{t=t_{cross}}}$. A power measurement which integrates the current through a voltage source from time t_{start} to t_{end} is easily expressed with $p_k(\tau) = u(\tau - t_{start}) - u(\tau - t_{end})$, where $u(t)$ is the unit step function. The Elmore delay of a signal (see for example [28]) can also be expressed as an integral. Expression of each measurement as a convolution integral is essential to the computation of sensitivities by the adjoint method.

Following the steps of the derivation in [2], Tellegen's theorem [32] is invoked on the nominal circuit and an adjoint circuit with the same topology, but arbitrary elements. Then Tellegen's theorem is again invoked on the perturbed circuit and the adjoint circuit. The difference between the two sets of resulting equations is integrated over the time period of simulation. Time is run backwards and appropriate choices of branch constitutive relations (BCRs) are made in the adjoint circuit to yield the generalized expression

$$\begin{gathered} \sum_I \int_{t_0}^{t_f} \left(-\delta v_I \hat{i}_I\right) dt + \sum_V \int_{t_0}^{t_f} \left(\delta i_V \hat{v}_V\right) dt = \\ \sum_C \left[C\hat{v}_C(t_0)\delta v_c(t_0) - \delta C \int_{t_0}^{t_f} \hat{v}_C \dot{v}_C\, dt\right] + \sum_R \delta R \int_{t_0}^{t_f} i_R \hat{i}_R dt + \\ \sum_{\text{other elements}} \cdots \end{gathered} \tag{18}$$

In writing (18), adjoint circuit quantities are represented with a caret ($\hat{}$) symbol and δ is used to represent perturbations in circuit values (not to be confused with the use of δ for time-domain Dirac functions). The terms on the right hand side have been shown only for the linear resistors and capacitors in the circuit, but the equation is valid only when summed over all the elements of the circuit. Note that $\hat{i}_I$, the current of independent current sources in the adjoint circuit and $\hat{v}_V$, the voltage of independent voltage sources in the adjoint circuit are the adjoint excitations that must be chosen in order to express the sensitivity function of interest. In general, by proper choice of adjoint circuit elements, (18) can be summarized as

$$\sum_{I_s} \int_{t_0}^{t_f} \left(-\delta v_I \hat{i}_I\right) dt + \sum_{V_s} \int_{t_0}^{t_f} \left(\delta i_V \hat{v}_V\right) dt = \sum_{l} \beta_l \delta x_l, \tag{19}$$

where δx_l represents the variation in the sensitivity parameters and β_l represents the corresponding convolutions. Now we will manipulate the left hand side of (19) to the form $\sum_{k=1}^{M} h_k \delta m_k$ by appropriate choices of the $\hat{i}_I$ and $\hat{v}_V$ waveforms as shown below.

$$\begin{array}{lll} \text{For a voltage measurement} & : \hat{i}_I(\tau) & = -\delta(\tau - t_{cross}) \\ \text{For a crossing time measurement} & : \hat{i}_I(\tau) & = \frac{\delta(\tau - t_{cross})}{\dot{v}_I|_{t=t_{cross}}} \\ \text{For a power measurement} & : \hat{v}_V(\tau) & = u(\tau - t_{start}) - u(\tau - t_{end}). \end{array} \tag{20}$$

Equations (20) give us the current and voltage excitations that would be applied one at a time if we were interested in finding the sensitivities of the individual measurements. Instead, we weight each of the waveforms of (20) by the corresponding h_k. Then substituting (20) into (19), we obtain

$$\delta\Phi = \sum_{k=1}^{M} h_k \delta m_k = \sum_{l} \beta_l \delta x_l. \tag{21}$$

Taking (21) to the limit as $\delta x_l \to 0$, the required sensitivities can be picked off as $\frac{\partial \Phi}{\partial x_l} = \beta_l$ in the course of a single adjoint analysis. Note that this result cannot be derived from simple superposition, since the adjoint circuit is a time-varying circuit.

3.4 Implementation of the adjoint Lagrangian formulation

Some special considerations that were taken into account during the implementation of the adjoint Lagrangian formulation are detailed in this section.

Sorting of pulses and impulses: Once the nominal simulation has been completed, the values of the measurements are known and the excitations for the adjoint circuit can be created. The pulses and impulses of the adjoint circuit are then 'quick-sorted' in reverse time order to be applied to the adjoint circuit in an event-driven manner, after being appropriately scaled. Event-driven techniques are used both to analyze the adjoint circuit and to carry out the necessary convolutions. In SPECS, the convolutions are typically between piecewise constant and either piecewise constant or piecewise linear waveforms (see [26, 27, 28]). To speed up the adjoint analysis, the time origin is shifted to the time of the first externally applied excitation to the adjoint circuit.

Multiple adjoint Lagrangian groups: In JiffyTune, typically a constraint or objective function includes a linear combination of observable circuit measurements. For example, the rise time of a transition can be expressed as the difference between the 10% and 90% crossing times of a node voltage. Thus, a single objective function or constraint of the optimization problem typically consists of a 'group' of individual measurements. 'Group adjoint' gradient computation is a procedure in which each such group of measurements is treated by LANCELOT as a scalar function, whose gradients can be computed by a single adjoint analysis. SPECS allows for multiple such 'adjoint Lagrangian groups'. The advantage is that the adjoint analysis is performed only as many times as the number of groups, rather than as many times as the number of individual measurements. As we will see below, multiple groups are necessary for computations of the Hessian, the matrix of second partial derivatives.

Scaling of the adjoint circuit excitations: Once the transient simulation has been completed in a new optimization iteration, the measurements values are fed to JiffyTune, while SPECS stands by for gradient computation. SPECS communicates to Jiffytune via a semaphore scheme. LANCELOT uses the measurement values to update its merit function and decides whether to accept the proposed step. If the step is rejected, gradient computation is skipped. If the step is accepted, LANCELOT provides to JiffyTune the values of the slack variables, Lagrange multipliers, penalty parameter and scale factors. As we have already seen, JiffyTune uses these values to determine the scaling of each pulse or impulse in the form

$$h_k = l_k + \sum_{i=1}^{M} n_{ki} m_i, \tag{22}$$

where h_k are the scale factors to be applied to each measurement, and l_k and n_{ki} are elements of a vector and sparse matrix that are created as a result of the chain ruling described in (14), (15) and (16).

4 Nonlinear optimization in JiffyTune

In the introduction we discussed various techniques used in circuit tuning, including geometric programming [10]. It is perhaps worth pointing out at this point that from the point of view of mathematical optimizers, geometric programming is an old technique whose development preceded the considerable advances in the field of nonlinear programming. Because of its inherent inflexibility it is rarely the most appropriate method. Furthermore, most circuit tuning problems as stated are not geometric problems. One can choose to transform the original problem into a geometric programming problem, but at the cost of (possibly considerable) inaccuracy and inflexibility.

Even for moderately sized circuit optimization problems, it is inappropriate to use genetic algorithms, simulated annealing, Nelder-Meade pattern search [33] or other heuristic methods when first derivatives are available. The optimization engine of JiffyTune is based upon the large-scale gradient-based nonlinear programming package LANCELOT. We give below a brief description of LANCELOT.

4.1 LANCELOT

The optimization engine of JiffyTune is based upon the large-scale nonlinear programming package LANCELOT. The kernel algorithm is an adaptation of a trust-region method to the general nonlinear optimization problem subject to simple bounds. The method is extended to accommodate general constraints by using an augmented Lagrangian formulation and the bounds are handled directly and explicitly via projections that are easy to compute. In the context of unconstrained optimization, trust-region methods, combining an intuitive framework with a powerful and elegant theoretical foundation, have led to robust numerical implementations. An excellent reference is [34]. The extension to problems with simple bounds is relatively straightforward and is illustrated in Figure 3 for a quadratic model function and an l_∞ trust-region. Essentially, global convergence can be guaranteed, provided one does at least as well as the generalized Cauchy point (w in the figure) that corresponds to the minimum along the projected gradient path ($x^k \to u \to w$) within the trust-region, where the projection is with respect to the bounds (either those provided by the user or implicit in the

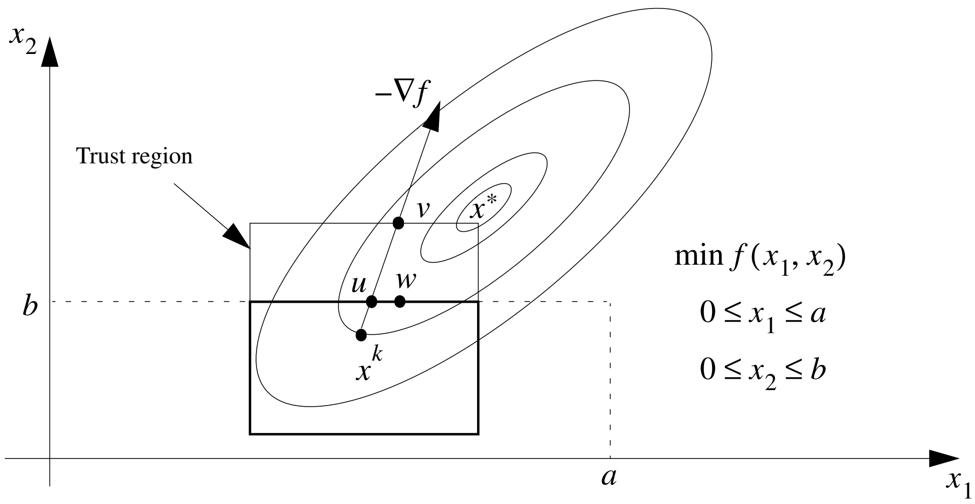


Figure 3: Illustration of the generalized Cauchy point.

trust-region). If a variable, as determined by the generalized Cauchy point, is at a bound it is said to be an activity. Unbounded variables are free. Activities are fixed temporarily, thus reducing the dimensionality of the search space (from two to one in the figure). Then, using only the free variables, the model of the objective function is further minimized within the feasible region and within the trust-region (w is optimal in Figure 3). Thus one obtains satisfactory asymptotic convergence. The function is evaluated at this point to determine how well the model predicted the actual change in the objective function. If good agreement is obtained, the approximate minimizer is accepted as the next iterate $\left(x^{k+1} \leftarrow x^k + s^k\right)$ and the trust-region is expanded. If only moderate agreement is obtained, the trust-region size remains unchanged, but the step is accepted. Otherwise, no new point is accepted and the trust-region is contracted. The beauty of such an approach is that, when the trust-region is small enough and the problem smooth, the approximation is necessarily good provided the gradients are sufficiently accurate. The above procedure has been proved to correctly solve the optimization problem under certain mild assumptions. Details are given in [19].

The extension to handle equality constraints is carried out by means of an augmented Lagrangian function that is minimized subject to the explicit bounds, using the earlier algorithm. Inequality constraints are converted to equality constraints by first introducing slack or surplus variables. We test for convergence by determining if we are sufficiently stationary (the projected gradient with respect to the simple bounds of the augmented Lagrangian is sufficiently small) and sufficiently feasible (the norm of the constraint violations is sufficiently small). Otherwise we use the minimization algorithm to find an approximate stationary point subject to the simple bounds. If this point is sufficiently feasible, we update the associated multipliers of the augmented Lagrangian function and decrease the tolerances for stationarity and feasibility. Otherwise, we give more weight to feasibility by decreasing the penalty parameter and resetting tolerances for stationarity and feasibility. It is possible to show, under suitable conditions, that convergence to a first-order stationary point for the nonlinear programming problem is attained. Further, if there is a single limit point, eventually the penalty parameter is not reduced. Details of these and other theoretical properties are given in [20] and [35].

The symmetric linear systems of equations that arise during the optimization are iteratively solved by using a preconditioned conjugate gradient method. A good description of conjugate methods is given in [36], Sections 4.8.3 and 4.8.5. The LANCELOT package offers several preconditioners, and the Schnabel-Eskow preconditioner [37] is used in JiffyTune. A detailed reference on the LANCELOT package, including all the available options, is given in the book [21] that accompanies the original software.

4.2 Application of LANCELOT to JiffyTune

In the context of JiffyTune it was necessary to make certain modifications to LANCELOT to account for the fact that the function and gradient values from SPECS, although accurate to within small perturbations, are noisy. The introduced errors are small but significantly larger than machine precision. Because of the complexity of general nonlinear optimization, many initializations (such as the choice of the initial trust-region radius or quadratic model) are based upon intelligent guesses, which cannot, of course, be ideal in all circumstances. In the worst case, for functions without noise, unfortunate choices can result in inefficiencies, but in the noisy case they can be insurmountable. For example, if the choice of initial trust-region radius is 0.001 μm, and a step of that size is well into the noise region of the problem, then the optimizer may never recover. For similar reasons, we had to introduce looser tolerances for feasibility, line search discontinuities and bound activities, replacing tolerances in the original software which were selected on the basis of machine precision. Finally, in order to stop gracefully and predictably, we needed to consider step sizes beneath which further progress is unlikely and relate stopping criteria to this step size in a robust and consistent manner.

Besides the ramifications of the adjoint Lagrangian implementation, two other LANCELOT enhancements deserve special mention. First, slack or surplus variables corresponding to satisfied inequalities are updated at each iteration so that the corresponding equality is satisfied exactly. In addition, we make the following observation: suppose that all the variables other than slack variables are temporarily fixed. The remaining augmented Lagrangian function with respect to the slack variables is a quadratic function and so can be optimized analytically. More importantly, in our context such optimization can be carried out without any further simulation. Whenever such an update is consistent with the resulting modified convergence theory the result has been a reduction in the number of iterations to convergence. Details are given in [38], where they are called 'two-step updates' (see also the spacer steps of [39]).

Second, minimax optimization is handled by the introduction of an additional linear variable and the reformulation of the problem as a general nonlinear programming problem. For example, suppose one had the problem

$$\underset{x \in \Re^n}{\text{minimize}} \quad \underset{i \in M \equiv \{1,2,\cdots,m\}}{\text{maximum}} \quad f_i(x). \tag{23}$$

This problem can be reformulated as

$$\begin{array}{ll} \underset{z \in \Re, x \in \Re^n}{\text{minimize}} & z \\ \text{subject to} & z - f_i(x) \geq 0, i \in M \equiv \{1,2,\cdots,m\}. \end{array} \tag{24}$$

Note that, as for slack/surplus variables, the two-step updating scheme mentioned above can also be exploited for the introduced minimax variable z.

JiffyTune also provides for design for manufacturability (DFM). A process corner consists of a set of device model parameters (e.g., temperature, power supply voltage, transistor length and width, bias) corresponding to a sample point of the process space created by inherent variations in manufacturing. The sampling is done so as to capture the extremes in the distribution of circuit behavior. For example, the parameter sets corresponding to typical, best case and worst case process corners are provided. We consider the following abstraction of the DFM problem: find the best assignment of transistor (and wire) widths so that the tuning problem is optimally solved across several process corners.

JiffyTune includes two DFM modes. The less sophisticated of these is to carry out a nominal tuning as usual, but in addition simulate the final tuned circuit once at each process corner supplied by the circuit designer. All measurements and functions are computed for each of these process corners. They are printed in a summary table so that the designer is able to understand the statistical variation of the performance of the circuit. In the more sophisticated DFM mode, JiffyTune simultaneously tunes at all process corners so that there are no post-tuning surprises. At each iteration, the circuit is simulated at each of the process corners provided, including computation of the gradients. This analysis takes k times as long if k process corners are to be considered. The optimizer simultaneously optimizes the circuit at all specified process corners so as to satisfy constraints and minimize objective functions across all corners. For example, an equality or inequality constraint is simply replicated at each process corner, an objective function is converted to a minimax function, so that the worst function across all the process corners is minimized and a minimax function is converted to a more general minimax function as follows: if the nominal problem is to minimize the worst of t functions, then the worst of tk functions is minimized in this mode, where k is the number of process corners being simultaneously considered.

4.3 Hessian computations in the adjoint Lagrangian formulation

In the non-adjoint Lagrangian formulation, the Hessians (matrices of second partial derivatives) of the objective function and each constraint are approximated using low-rank updates. This method requires the gradients of the objective function and of the constraints, which are not available in the adjoint Lagrangian formulation. Consequently, Hessian computations in the adjoint Lagrangian formulation need to be treated differently.

4.3.1 Hybrid scheme for the computation of the Hessian in the adjoint Lagrangian formulation

LANCELOT uses a quadratic model of the merit function at each iteration. Its Hessian matrix is built up by low-rank quasi-Newton update methods. For a quadratic function $f(x)$, the equation $\nabla^2 f(x)d = \nabla f(x+d) - \nabla f(x)$ is satisfied exactly for any direction d. Analogously, the quasi-Newton condition maintains an approximate Hessian, B, that satisfies $Bd = \gamma$,where γ is the appropriate gradient difference. The idea of a low-rank update (changing B by a matrix whose columns span a space of low dimension) is to modify B with new gradient difference (or approximate curvature) information at moderate cost while maintaining the quasi-Newton condition. Low-rank updates are desirable since they make the accompanying linear algebra relatively inexpensive. In 'minor iterations' of LANCELOT, only the problem variables and slacks change. In 'major' iterations, usually terminated by sufficient stationarity, either the penalty parameter or the Lagrange multipliers change, depending on whether sufficient feasibility has been achieved or not during the inner unconstrained problem. The merit function that LANCELOT builds is

$$\Phi(x) = f(x) + \sum_i \lambda_i c_i(x) + \frac{1}{2\mu} \sum_i c_i^2(x), \tag{25}$$

where f is the objective function and c_i are the constraints. Hence

$$\nabla\Phi(x) = \nabla f(x) + \sum_i \lambda_i \nabla c_i(x) + \frac{1}{\mu} \sum_i c_i(x) \nabla c_i(x), \tag{26}$$

and the Hessian

$$\begin{aligned} \nabla^2\Phi(x) \; &= \nabla^2 f(x) + \textstyle\sum_i \lambda_i \nabla^2 c_i(x) \\ &\quad + \tfrac{1}{\mu} \textstyle\sum_i \left(c_i(x) \nabla^2 c_i(x) + \nabla c_i(x) \nabla c_i(x)^T \right). \end{aligned} \tag{27}$$

As an initial approximation of $\nabla^2\Phi(x)$ at the outset of the optimization, $\nabla^2 f$ and $\nabla^2 c_i$ are taken to be zero matrices (except for the slack/surplus variables or the minmax z variables. See Section 4.3.2.) and $\nabla c_i(x)\nabla c_i(x)^T$ is computed explicitly. At each minor iteration, and at the start of each major iteration where λ is changed, low-rank quasi-Newton updates are used on the approximate composite Hessian matrix, B_k. However, every time a major iteration begins for which μ is changed, we take as our approximate Hessian $B_{k+1} = B_k + \left(\frac{1}{\mu_{new}} - \frac{1}{\mu_{old}}\right) \sum_i \left(\nabla c_i(x) \nabla c_i(x)^T\right)$. Hence, at the start of the optimization and at the start of each major iteration for which μ has changed, the gradient vector of each individual constraint is required. Since each constraint may in turn depend on multiple measurements, multiple 'adjoint Lagrangian groups' are used in SPECS. Thus a hybrid overall scheme is employed wherein adjoint Lagrangian gradient computation with just one group is used at every minor iteration or major iteration for which sufficient feasibility was attained, but an adjoint analysis with as many groups as the number of constraints is used at the start of each major iteration otherwise. Although the latter is more expensive, the reduction in iterations due to starting the major iteration with a better Hessian approximation more than compensates for the added computational cost.

4.3.2 Hessian updates with respect to slack variables in the adjoint Lagrangian formulation

By taking advantage of the form in which slack variables occur in the augmented Lagrangian, the Hessian entries with respect to slack variables can be explicitly computed. However, using explicitly computed values violates the quasi-Newton condition [36] when standard Hessian update formulas are used. Hence, we have developed modified update formulas which satisfy both the quasi-Newton condition and allow us to assign explicitly computed Hessian entries.

The formula is demonstrated below for the case of a problem with an objective function and just one inequality constraint. Then

$$\Phi = f(x) + \lambda(c(x) + s) + \frac{1}{2\mu}(c(x) + s)^2, \tag{28}$$

where s is the slack variable. Clearly,

$$\nabla_s^2 \Phi = \frac{1}{\mu}. \tag{29}$$

If (29) is applied after the regular Hessian updates, the quasi-Newton condition will be violated. Instead, we introduce the rank-two update

$$\begin{aligned} B_{k+1} = \; & B_k + \tfrac{1}{v_k^T E s_k} \left[(y_k - B_k s_k)\, v_k^T E + E v_k \,(y_k - B_k s_k)^T \right] - \\ & \tfrac{(y_k - B_k s_k)^T s_k}{\left(v_k^T E s_k\right)^2} E v_k \,(E v_k)^T , \end{aligned} \tag{30}$$

where at the i^{th} iteration, B_i is the Hessian approximation, y_i is the change in $\nabla\Phi$, s_i is the step and E is a diagonal matrix with a zero on the diagonal corresponding to each slack variable and one otherwise. This novel update formula preserves the quasi-Newton condition. If the slack entries are correctly set in B_0, then the update formula (30) guarantees that they remain correctly set after the update. By choosing v_k as one of $(y_k - B_k s_k)$, s_k and y_k, we obtain modified SR1 (Symmetric rank-one), PSB (Powell-symmetric-Broyden) and DFP (Davidon-Fletcher-Powell) Hessian updates, respectively. By adding the term $-(s_k^T E B_k s_k) w_k w_k^T$, where $w_k = E\left(\frac{1}{y_k^T E s_k} y_k - \frac{1}{s_k^T E B_k s_k} B_k s_k\right)$, to the modified DFP update, we obtain the modified BFGS (Broyden-Fletcher-Goldfarb-Shanno) update [36]. One can similarly treat the explicitly available Hessian terms corresponding to the additional variable z introduced in Section 4.2 to handle minimax problems.

5 JiffyTune interface and environment

The JiffyTune engine as reviewed in Section 2 is driven by a textual control file that describes the optimization problem. From the inception of the JiffyTune project, it was realized that generation of such files should be automated as much as possible and that a good human interface and an intuitive abstraction of its use and behavior would be crucial to acceptance of the tool by circuit designers. Interfaces were built to run the tool from the Cadence [40] and SLED [41] schematic design systems. Integrating the tool into such a framework capitalizes on the familiarity of the user with the schematic design environment, and lends a visual and interactive aspect to the tool. Many of the complexities are hidden from the designer, although care was taken to allow full access to all tool functions, if the designer so requires. The basic functions of the interface are listed below.

Specification of tuning parameters: Tunable transistors are specified simply by selecting transistors or gates on a hierarchical schematic. The tunable transistors/gates are visually marked by a flag to indicate tunability. Facilities are provided to ratio transistors. Transistor ratio-ing is of particular importance for dynamic logic, for example to maintain a specific ratio between the half latch and evaluate devices in domino-type logic, or to maintain transistor ratios in a tapered stack while tuning. In addition, similar instances (transistors, gates or higher-level functional blocks) can be 'grouped' together, to ensure that corresponding transistors in those blocks track during tuning.

Specification of measurements and functions: Presently, the interface supports delay, transition time (slew), area and power functions. For delay and transition times, net selection is done directly on the schematic. Power functions are specified by selecting the required voltage source, again directly on the schematic. In all cases, the user is prompted to provide a relation and target value as described in Section (2.1). In the schematic environment, with no knowledge of layout, area targets are approximated by the sum of the widths of the tunable transistors and wires. The appropriate linear combination of measurements is written to the control file in each case. Minimax functions can be defined over any set of existing measurements.

Specification of controls: Administrative information such as the maximum number of iterations, location of device model files, process corners information, desired method for computing the sensitivities and the layout grid for rounding transistor widths at the end of optimization can be specified in a form that is pre-filled with project-specific defaults.

Execution of JiffyTune: After specifying parameters, functions and controls, the designer can ask for all this information to be written to a control file. Then the designer can launch the JiffyTune engine, whereupon the progress of the optimization is displayed.

Back-annotation of the results: The results of a JiffyTune run are back-annotated onto the schematic as suggested transistor widths next to transistors (or as new parameters next to gates). The designer can then accept these new widths/parameters, selectively or as a whole. Further, a facility is provided to back-annotate final waveform characteristics, such as delay through a gate or rise time of a net, directly onto the schematics, relieving the designer of the need to browse through simulation data using a waveform viewer.

Utilities: The JiffyTune menu also includes facilities replicated from other areas of the schematic design environment, such as schematic saving, netlisting and automatically adjusting the number of fingers on each transistor, to create a single integrated tuning environment.

Circuit requirements must be specified with care, since the optimizer will take advantage of any unspecified aspects. For example, area minimization will shrink to its minimum size a transistor that does not contribute materially to any measured transition. Thus, the tool enforces clear expression of circuit requirements that otherwise are often tacit. Since these circuit requirements and attributes logically belong with the circuit (they are indeed part of the intellectual effort of designing the circuit), the tuning parameters and functions are stored in the design database, either as instance properties (tunability, upper and lower bounds on transistor widths) or as schematic properties (grouping, functions). This practice also encourages the reuse of circuits; if a tuning problem has been adequately specified, the circuit can easily be retuned.

6 Computational results

6.1 Sensitivity benchmarks

We report results for the direct method, measurement-at-a-time adjoint approach and the adjoint Lagrangian technique, as implemented in JiffyTune.

Run time	**Direct method**	**Adjoint method**	**Adjoint Lagrangian method**
Total run time	22.13	12.41	2.42
Run time for sensitivity computation only	19.96 (168 meas.)	10.38 (36 meas.)	0.37
Run time per sensitivity circuit solution	0.119	0.288	0.37
Run time per sensitivity circuit solution as a fraction of simulation time (2.03 seconds)	5.86%	14.19%	18.05%
Run time per gradient computation	3.3×10^{-3}	1.72×10^{-3}	6.12×10^{-5}
Run time per gradient evaluation as a fraction of simulation time	0.16%	0.085%	0.003%

Table 1: Sensitivity computation run time.

A dynamic logic 'branch scan' circuit with 144 MOSFETs, an actual circuit from a high-performance PowerPC microprocessor, was chosen to demonstrate the efficiency of the gradient computation. The circuit was simulated in SPECS for a simulation interval of 27 ns.

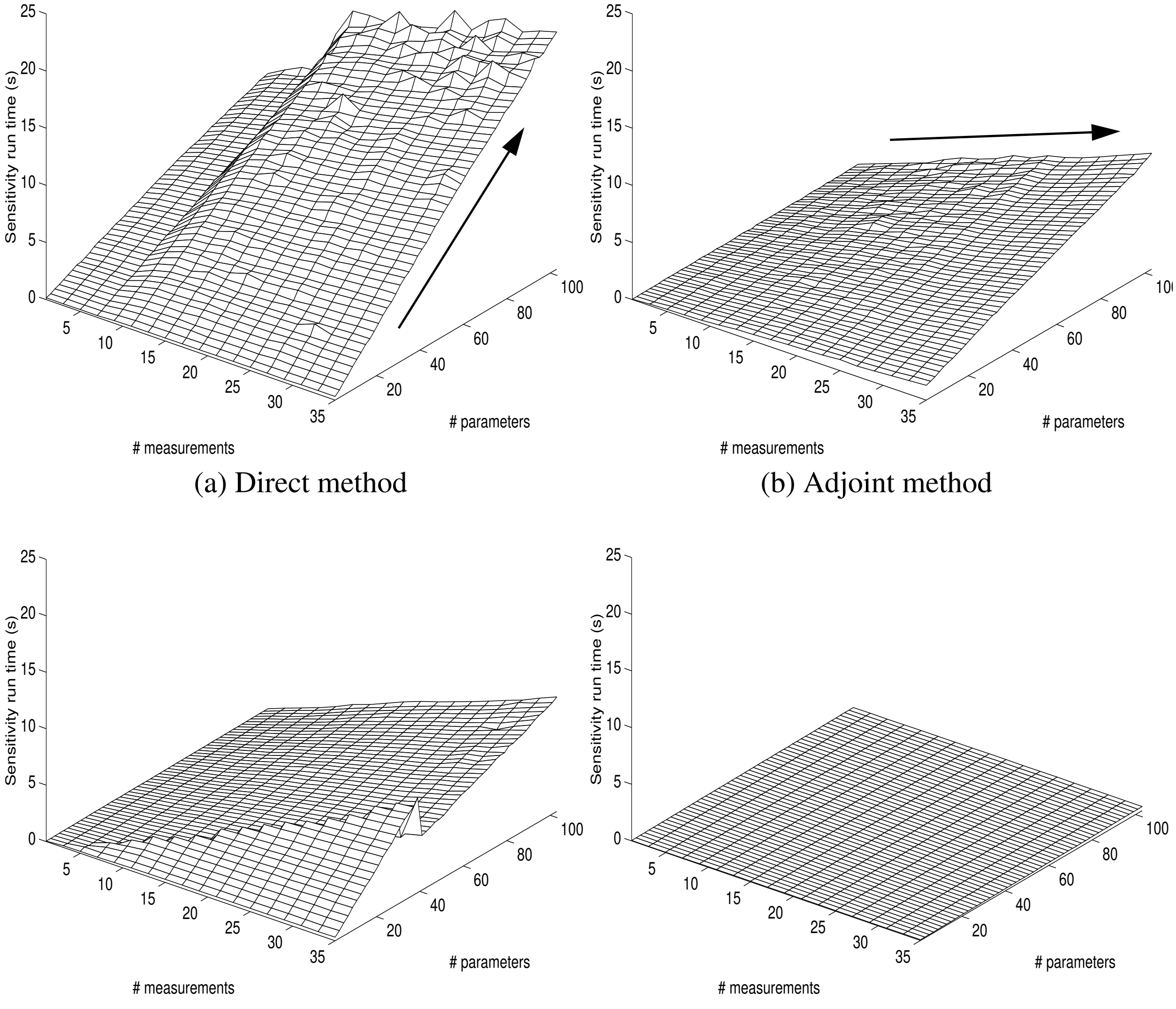
The CPU time for simulation was 2.03 seconds on an IBM Risc/System 6000 model 590. Then the same simulation run was carried out with 36 sensitivity functions (crossing times) and 104 MOS transistor widths as sensitivity parameters. Since there were 64 diffusion and other parasitic capacitances dependent on these 104 transistor widths, the total number of sensitivity parameters was 168. The number of gradients computed in this benchmark was $6,048$, since SPECS finds the gradient of every sensitivity function with respect to each sensitivity parameter (our Jacobian matrix is dense). The run times of SPECS with all three sensitivity computation methods (direct, adjoint and adjoint lagrangian) on this benchmark circuit are shown in Table 1. From the table, we see that the total run time for a JiffyTune iteration would be 2.42 seconds (assuming that the adjoint Lagrangian method were used). For comparison, the AS/X [24] run time on this circuit (with no gradient computation, of course) was 40.11 seconds. Hence, even on this modest example, JiffyTune can complete seventeen iterations with gradient computation in the time it takes AS/X to simulate the nominal circuit once.

As can be seen from the table, the overhead of computing one gradient is a fraction of a percent of the original simulation time, which works out to 61 microseconds or less of CPU time in this example. The overhead of one sensitivity circuit analysis is about 5.9% for the direct method and about 15% for the adjoint method. Note that the number of runs in the adjoint method is equal to the number of functions, while it is equal to the number of sensitivity parameters in the direct method and is equal to one in the adjoint Lagrangian method. The higher overhead in the adjoint method is accounted for by the convolution required between the waveforms of the original circuit and the sensitivity circuit.

Next we tested gradient computation on the same dynamic branch-scan-select circuit while varying the number of measurements from 1 to 36 and the number of tunable transistors from 1 to 104. Four analyses were conducted for each resulting combination. In the first analysis, the direct method of sensitivity computation was used. The run time of sensitivity computation as a function of the number of measurements and parameters is shown in Figure 4(a).

As can be seen from the figure, the incremental cost of each additional sensitivity parameter is quite high (see the bold arrow in Figure 4(a)), as predicted by the theory. Figure 4(b) shows the run time using the adjoint method, computing the gradients of individual measurements. Again, as indicated by the bold arrow, the growth of run time with each additional measurement is quite high. In Figure 4(c), the run time of our previous production version is shown, in which a heuristic is used to pick the sensitivity analysis method. If the parameters outnumber the measurements by more than a factor of 3, the adjoint method is chosen. The figure clearly shows a ridge where the program switched from the direct to the adjoint method. Finally, Figure 4(d) shows the run time of the adjoint Lagrangian formulation wherein a single adjoint analysis was used to compute the gradients of the merit function with respect to all parameters. Figure 4(d) clearly demonstrates not only the speedup obtained by the novel method, but also the relatively slow growth of run time with respect to the number of parameters and almost imperceptible growth with respect to the number of measurements.

The adjoint Lagrangian formulation was tested on a set of 18 benchmark circuits, whose characteristics are shown in Table 2. An adjoint sensitivity analysis was performed on each of these 18 benchmark circuits to compute the gradients of individual measurements. Then the sensitivity analysis was repeated with an adjoint Lagrangian formulation, using a set of weights to form a linear combination of the measurements, as shown in (16). The gradients of the former analysis were combined in a post-processing step, using



(a) Direct method

(b) Adjoint method

(c) Heuristic choice of method

(d) Adjoint Lagrangian formulation

Figure 4: Runtime of gradient computations plotted against number of measurements and parameters.

Name	Number MOSFETS	Number Independent parameters	Number Dependent parameters	Number Measurements	Number Constraints	Objective function?	Minimax problem?
lau2	24	4	16	3	1	Y	N
morrill	8	3	3	2	1	N	N
davies3	235	15	87	2	1	Y	N
durham2	204	11	2	4	2	N	N
wire	8	8	3	4	0	Y	N
Nov01power	17	4	0	3	1	Y	N
fleischer	228	104	80	5	5	Y	Y
clkgen	28	17	10	6	5	N	N
Nov01	17	4	0	6	4	N	N
northrop_xor	15	9	2	16	8	Y	Y
coulman_delay	70	16	48	33	17	N	N
coulman_hot	70	16	48	33	17	N	N
coulman_cold	70	16	48	33	17	N	N
coulman_delay _minmax	70	16	48	33	17	Y	Y
coulman_hot _minmax	70	16	48	33	17	Y	Y
coulman_cold _minmax	70	16	48	33	17	Y	Y
bultmann	80	24	0	26	13	Y	Y
IOmux	6,900	60	4,068	82	41	Y	Y

Table 2: Characteristics of benchmark circuits.

the same weights, to compose the gradients of the composite merit function. The two sets of gradients were then compared. Across all the benchmarks, a total of 707,677 gradients were compared. The worst inaccuracy among all these gradients between regular adjoint analysis and adjoint Lagrangian analysis was 2.96×10^{-11} (in units of either ns/μm or mW/μm), demonstrating that the adjoint Lagrangian formulation does indeed produce the same results.

The run times and speedups for gradient computation only and for nominal simulation combined with gradient computation are shown numerically in Table 3 and graphically in Figure 5.

All CPU times in this paper are on an IBM Risc/System 6000 model 590 workstation. While Table 2 shows all measurements as defined by the user, the number of measurements in Table 3 and Figure 5 are the number of 'meaningful' measurements, discounting delay measurements on primary inputs whose gradients are known to be zero. A speedup of up to 36x in gradient computation is observed on circuits with a large number of measurements, which in turn leads to a speedup of up to 4.2x per iteration of JiffyTune. Figure 5 shows the speedup of simulation combined with gradient computation, speedup of just the gradient computation and the number of non-trivial measurements in each benchmark. From the theoretical discussion of Section 3.3, the number of measurements is an upper bound on the practically achievable speedup. Note that on some of the smaller examples, a speedup higher than the theoretically predicted speedup is achieved due to the granularity of CPU time measurements. We remark that the faster gradient computation renders feasible the optimization of much larger circuits than was previously possible.

6.2 Case study of JiffyTune use

JiffyTune was applied to tune custom circuits in the critical paths of a high-performance, dynamic-logic PowerPC microprocessor. The circuits consisted of a mix of transistors and continuously parameterized gates. The Cadence graphical user interface made it possible for designers to use the tool with little or no training. JiffyTune was used by 58 designers during about 2,269 interactive sessions to tune 388 unique circuits. Over 6,289 successful JiffyTune runs were carried out, showing that some circuits were re-tuned multiple times. The results of tuning on one particular benchmark circuit using the measurement-at-a-time formulation are presented below. Table 4 lists the results of running JiffyTune on a 12-way priority decode circuit under four different conditions. The circuit contains 70 MOSFETs

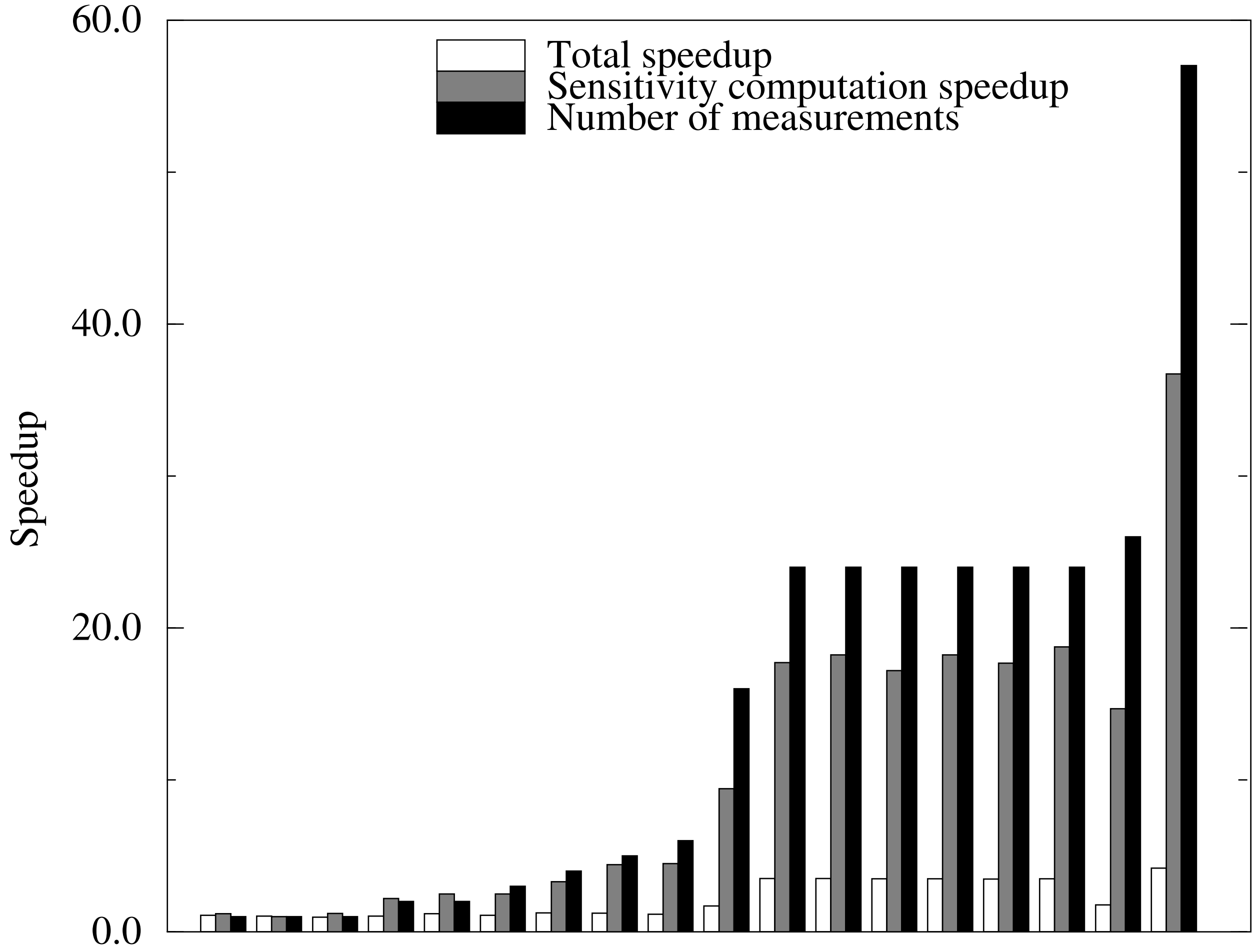


Figure 5: Histogram of speedups for the 18 benchmark circuits.

Name	Number MOSFETS	Number Measure -ments	Number parameters	Sensitivity computation time (CPU seconds)			Total run time (CPU secs.)		Total Speedup factor
				Adjoint	Adjoint Lagrangian	Speedup factor	Adjoint	Adjoint Lagrangian	
lau2	24	1	20	0.12	0.1	1.2	1.42	1.3	1.1
morrill	8	1	6	0.01	0.01	1.0	0.27	0.26	1.0
davies3	235	1	102	0.58	0.48	1.2	12.4	12.8	0.97
durham2	204	2	13	0.46	0.21	2.2	5.86	5.63	1.0
wire	8	2	11	0.05	0.02	2.5	0.51	0.43	1.2
Nov01power	17	3	4	0.05	0.02	2.5	0.47	0.43	1.1
fleischer	228	4	184	1.91	0.58	3.3	5.98	4.77	1.3
clkgen	28	5	27	0.31	0.07	4.4	1.3	1.05	1.2
Nov01	17	6	4	0.09	0.02	4.5	0.51	0.44	1.2
northrop_xor	15	16	11	0.66	0.07	9.4	1.83	1.07	1.7
coulman_delay	70	24	64	6.2	0.35	17.7	8.15	2.32	3.5
coulman_hot	70	24	64	6.2	0.34	18.2	8.14	2.32	3.5
coulman_cold	70	24	64	6.19	0.36	17.2	8.12	2.32	3.5
coulman_delay _minmax	70	24	64	6.2	0.34	18.2	8.14	2.33	3.5
coulman_hot _minmax	70	24	64	6.19	0.35	17.7	8.12	2.34	3.5
coulman_cold _minmax	70	24	64	6.19	0.33	18.8	8.13	2.33	3.5
bultmann	80	26	24	1.91	0.13	14.7	4.03	2.27	1.8
IOmux	6,900	57	4,128	690	18.8	36.7	882	210	4.2

Table 3: Sensitivity computation and total speedups.

and the simulation was run for 35 ns. The tuning runs all had 64 tunable transistors, of which 16 were independent and 48 dependent. The 17 functions to be optimized included the rising delay through four critical paths, the falling delay through those paths, the rise/fall times on each of the above 8 transitions and an area constraint. For confidentiality purposes, the delay requirement on the worst of the critical paths has been normalized to 500 time units for the table on this benchmark. The table lists the rising and falling delay of the four paths being tuned as predicted by AS/X on the final design (the worst of the 8 delays for each run is shown in bold), the total tunable transistor area of the circuit and the total CPU time required to run JiffyTune on an IBM Risc/System 6000 model 590. The first JiffyTune run (HOT) started from a circuit that had previously been manually tuned (MANUAL). The worst delay through the circuit improved by 7.5% and the area decreased by 5.0%. The second JiffyTune run (COLD) started from an untuned circuit in which initial transistor sizes were set to the same default value as they would be for a 'new design.' Comparing the results of (HOT) and (COLD) shows that the poor start point did not significantly change the final results, but the optimizer had to work harder. The next JiffyTune run (DELAY) was set up to cause JiffyTune to reach the timing goal of 500 time units at all cost. JiffyTune was configured as in run (HOT), only with a weight on the area constraint that was a tenth of the previous value. The table shows that the goal was reached but at a high cost in transistor area. In general, we have found that it is important to impose an area constraint, since without an area constraint, JiffyTune converges to one of many equally fast circuits depending on the start point, with some solutions more compact than others. The final run used the same start point and weights as HOT, but formulated the problem as a minimax optimization. A solution with a somewhat higher delay but lower area was obtained in this case.

6.3 Circuit optimization

The benchmark circuits of Table 2 were optimized using the adjoint Lagrangian formulation and the new method of Hessian updates. However, unlike the sensitivity comparisons of Section 6.1 the non-adjoint Lagrangian results presented here included enhancements such as the multiple adjoint Lagrangian groups used in the hybrid scheme when the multiplier changed at a major iteration and gradient skipping for rejected steps. A comparison of results on the benchmark problems is given in Table 5. The speedups shown in the last column of Table 3 are further eroded during the optimization procedure due to this hybrid scheme being necessary for such major

	MANUAL	**HOT**	**COLD**	**DELAY**	**MINIMAX**
Path #1, falling delay	555	494	488	483	497
Path #1, rising delay	471	475	473	469	510
Path #2, falling delay	535	495	495	483	506
Path #2, rising delay	494	488	488	472	524
Path #3, falling delay	**561**	**519**	517	**497**	**544**
Path #3, rising delay	497	**519**	**519**	**497**	527
Path #4, falling delay	497	494	491	484	516
Path #4, rising delay	462	497	496	485	476
Area	893	844	849	1148	800
# JiffyTune iterations	-	9	26	16	41
Run time (CPU seconds)	-	172	465	289	716

Table 4: JiffyTune results for 12-way priority decode circuit; all delays are normalized to a requirement of 500 time units.

iterations in the adjoint Lagrangian scheme, in addition to various other overheads such as reading and compiling files in the process of setting up the optimization problem, the optimization routines and further tasks that are common to the old and new implementations. Nevertheless, some gains in CPU time were observed with no significant loss in the quality of results in almost all cases. The difference in the final result between adjoint Lagrangian and non-adjoint Lagrangian tuning is due to the Hessian being approximated differently, resulting in different paths being taken by the optimization program. The real benefit of the adjoint Lagrangian formulation is seen in larger problems, particularly problems with a large number of measurements.

In all cases except the IOmux benchmark the problems were run until completion. However, in the case of IOmux the stopping criteria were not readily satisfied and the runs continued for more than double the number of iterations without improvement. Since this obscures the results, we report IOmux after 27 iterations for both adjoint and non-adjoint (when a best solution had already been found). Meanwhile we are seeking ways to improve the stopping criteria. Some more detailed comments on the results of optimizing the IOmux circuit of Tables 2 and 3, are presented below.

[1] On problems with only constraints, the optimizer was instructed to stop if the sum of the constraint violations was under 2 ps. Hence any value under 2ps means that the problem was essentially solved.

Name	Non-Adjoint Lagrangian value (ps)	Adjoint Lagrangian value (ps)	Function/Gradient/Iterations (CPU secs./ CPU secs./ #) Adjoint	Function/Gradient/Iterations (CPU secs./ CPU secs./ #) Adjoint Lagrangian	Speedup factor Gradient	Speedup factor Total
lau2	164	161	27.10/1.59/25	52.80/2.94/50	0.54	0.54
morrill[1]	0.168	0.526	2.83/0.07/9	3.32/0.10/10	0.70	0.92
dav3	256	264	314.00/5.53/25	409.00/11.20/34	0.49	0.76
dur2	472	472	91.50/1.30/17	90.50/0.92/17	1.41	1.01
wire	15700	13600	13.70/1.15/33	38.60/2.72/100	0.42	0.41
Nvpw	267	262	12.50/0.60/29	42.10/1.20/100	0.50	0.35
fleis	-532	-511	353.00/86.70/88	328.00/38.50/82	2.25	1.19
clkgen[1]	1.73	1.54	5.97/0.72/5	12.40/0.77/12	0.94	0.62
Nov01	176	176	15.20/1.11/35	18.70/0.56/43	1.98	0.86
nort	-25.4	-2.47	43.10/8.31/47	25.30/2.22/28	3.74	1.76
co_d	119	122	42.70/34.50/22	54.60/9.48/28	3.64	1.19
co_h	270	290	38.20/37.30/19	35.70/6.35/17	5.87	1.73
co_c	261	263	34.50/20.60/17	50.40/8.50/25	2.42	0.94
c_d_m	73.8	93.9	98.20/127.00/51	98.50/28.90/52	4.39	1.75
c_h_m	75.8	109	57.40/66.60/28	57.00/19.00/27	3.51	1.60
c_c_m	72.5	77.6	157.00/178.00/80	191.00/44.30/99	4.02	1.42
bult	-71.4	-61	48.20/9.13 /24	25.20/3.14/13	2.91	1.91
IOmux	-14653	-11625	5137.86/7965.87/27	4992.76/1504.62/27	5.29	2.02

Table 5: Overall comparison of adjoint Lagrangian results.

The IOmux circuit tuning problem was formulated as an area minimization with 41 timing constraints and a relatively high weight on the area objective function. The area (approximated by the sum of the transistor widths) began at $31,128\ \mu$m. The non-adjoint Lagrangian method reduced the area to $14,044.8\ \mu$m in the course of 26 optimization iterations but the sum of the infeasibilities was $18,340$ equivalent ps. After 27 iterations the total CPU time required was 218.4 minutes, consisting of 85.6 minutes of transient simulation and 132.8 CPU minutes of gradient evaluation time. With the adjoint Lagrangian formulation, after 27 iterations the area was $15,185.3\ \mu$m but the sum of the infeasibilities was $12,451$ equivalent ps.

The run time reduced to a total of 108.3 minutes, consisting of 83.2 minutes of transient simulation CPU time and 25.1 minutes of gradient evaluation time. Thus, the overall speedup in the optimization was 2.02, while the speedup in the gradient computation portion was 5.29. Clearly the gradient computation bottleneck has been effectively addressed, leaving the transient simulation as the dominant portion of the total run time. We remark that the optimization routines in the adjoint Lagrangian run occupied a paltry 11 seconds of CPU time. The gradient speedup would be equal to the value reported in Table 2 (i.e., 36.7 rather than 5.29) if a composite adjoint Lagrangian formulation were used in every iteration. However, in the hybrid scheme, a 'group' adjoint Lagrangian formulation is used for some of the major iterations. Hence the hybrid scheme reduces the total gradient computation speedup.

In the above example, the adjoint Lagrangian formulation reduced the CPU time of the circuit optimization from over 218 minutes to under 108.5 minutes. This speedup is expected to improve further as the method is applied to larger circuits, thus rendering such optimizations feasible [2]. We have recently successfully tuned a circuit with $18,854$ transistors, $12,608$ tunable transistors and 6 independent parameters in less than two hours. Further, the adjoint Lagrangian formulation allows additional constraints at a relatively low incremental cost. In particular, increasing the number of independent parameters does not affect the sensitivity computations at all. It does result in a larger problem for the LANCELOT optimization engine, but the run time for LANCELOT is not a bottleneck. This feature has a significant methodology impact, particularly for self-timed and dynamic circuits in which the number of timing 'checks' that have to be satisfied

[2] Indeed our latest results produced an area of $15,669.2\ \mu$m after 15 iterations and 61.57 minutes with the sum of infeasibilities being $11,651$ ps and terminated on its own at iteration 23 and a further 32.55 minutes of run time.

during tuning can be very large. In many cases, the independent parameters are picked based on ratio-ing and grouping that the designer chooses in order to make the subsequent layout task manageable and the circuit "balanced." Less grouping implies a larger number of independent parameters, more freedom for the optimizer, often a better result, but more layout work since fewer cells can share layouts.

7 Conclusions and future work

In this paper we described JiffyTune, a program that optimizes circuits by adjusting transistor and wire sizes. JiffyTune makes use of fast simulation and time-domain gradient computation in the circuit simulator SPECS, and advanced nonlinear numerical techniques in the optimization package LANCELOT. Delay, rise/fall time, area and power optimization have been implemented. The optimization system is flexible and allows ratioing of transistors and grouping of identical instances. An intuitive interface including back-annotation of optimization results onto the schematic has been developed.

The environment in which a circuit will be used and the required performance are estimated long before the chip is built. By the time the circuit is integrated onto the chip, it may no longer be optimally tuned, much to the frustration of the design engineer. Changes in loading, changes in the specifications, changes in parasitics after extraction, changes in technology device models and remapping to a new technology are common occurrences during the course of a project. In such situations, retuning at the push of a button without tedious re-specification is extremely useful.

With the proposed adjoint Lagrangian formulation for the computation of circuit gradients, for the purposes of optimization, the gradients of a merit function can be computed in a single adjoint analysis, irrespective of the number of parameters or the number of measurements. Speedups of over 30x were demonstrated in the gradient computation procedure, thus addressing the bottleneck in circuit optimization programs. This approach to gradient computation has resulted in circuits with up to $6,900$ transistors being successfully tuned in about 108 minutes of CPU time. The largest circuit tuned contains $18,854$ transistors. The low incremental cost of additional constraints makes the optimizer amenable to tuning dynamic circuits, which typically have a large number of timing constraints. Improved methods for performing Hessian updates and better stopping criteria are currently being investigated to enhance the efficiency of the circuit optimization.

JiffyTune has been successfully used to tune a number of circuits on the critical paths of a high-performance microprocessor chip which makes liberal use of dynamic logic. It has been particularly useful in tuning tricky pass-gate circuits and has been found to enhance design re-use. Further, since the optimization process has been made relatively easy and automatic for the designer, a paradigm shift has been observed; the issue becomes how to correctly specify the optimization problem rather than solving the optimization problem itself. There are a number of avenues for future work. Repeated solution runs of the sensitivity or adjoint circuit are independent and therefore amenable to parallel processing. Extension to semi-infinite constraints (see, for example [42, 15]) would allow optimization of circuits while taking into account environmental variations such as temperature and power supply voltage. Reformulating the problem to take advantage of group partial separability in LANCELOT [21, 43] would speed up the optimization. In addition, applications to IC manufacturability are being considered.

We note that JiffyTune in its present form is not directly applicable to designs in which gates are chosen from a fixed library of cells with a finite set of discrete power levels. On the other hand, JiffyTune performs well on hierarchical schematics with leaf cells containing any mix of transistors and continuously parameterized gates. In practice, JiffyTune handles circuits containing pass transistors well, in contrast to optimizers based on static timing analysis, since SPECS yields electrically true sensitivities, taking into account details of the device model such as body effect. JiffyTune makes it possible to speed up the design process, make more refined designs and provide better information about performance trade-offs.

8 Acknowledgments

The authors would like to thank David Ling and the anonymous reviewers for their useful suggestions that contributed to the present manuscript.

References

[1] D. A. Hocevar, P. Yang, T. N. Trick, and B. D. Epler, "Transient sensitivity computation for MOSFET circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-4, no. 4, pp. 609–620, 1985.

[2] S. W. Director and R. A. Rohrer, "The generalized adjoint network and network sensitivities," *IEEE Transactions on Circuit Theory*, vol. CT-16, no. 3, pp. 318–323, 1969.

[3] G. D. Hachtel and R. A. Rohrer, "A variational approach to the optimal design and synthesis of switching circuits," *IEEE Proceedings*, vol. 55, no. 11, pp. 1864–1877, 1967.

[4] G. D. Hachtel, R. K. Brayton, and F. G. Gustavson, "The sparse tableau approach to network analysis and design," *IEEE Transactions on Circuit Theory*, vol. CT-18, pp. 101–113, 1971.

[5] J. P. Fishburn and A. E. Dunlop, "A posynomial programming approach to transistor sizing," in *IEEE International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[6] D. Marple, "Transistor size optimization in the TAILOR layout system," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 43–48, 1989.

[7] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM Journal of Research and Development*, vol. 1, pp. 100–105, 1982.

[8] W. C. Elmore, "The transient analysis of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, 1948.

[9] P. Penfield and J. Rubinstein, "Signal delay in RC tree networks," in *Proceedings of the 2nd Caltech VLSI Conference*, pp. 269–283, 1981.

[10] R. Duffin, E. Peterson, and C. Zener, *Geometric Programming - Theory and Applications.* New York: John Wiley & Sons, 1967.

[11] S. S. Sapatnekar and W. Chuang, "Power vs. delay in gate sizing: conflicting objectives?," in *IEEE International Conference on Computer-Aided Design*, pp. 463–466, 1995.

[12] F. Najm, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and System*, vol. CAD-9, pp. 439–450, 1990.

[13] J. J. Cong and C. Koh, "Simultaneous driver and wire sizing for performance and power optimization," *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, pp. 408–425, 1994.

[14] N. Menezes, R. Baldick, and L. T. Pileggi, "A sequential quadratic programming approach to concurrent gate and wire sizing," in *IEEE International Conference on Computer-Aided Design*, pp. 144–151, 1995.

[15] W. Nye, D. C. Riley, A. Sangiovanni-Vincentelli, and A. L. Tits, "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-7, pp. 501–519, 1986.

[16] J. Shyu and A. Sangiovanni-Vincentelli, "ECSTASY: a new environment for IC design optimization," in *IEEE International Conference on Computer-Aided Design*, pp. 484–487, 1988.

[17] C. Visweswariah and R. A. Rohrer, "Piecewise approximate circuit simulation," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 248–251, 1989.

[18] C. Visweswariah and R. A. Rohrer, "Piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuitss and Systems*, vol. 10, no. 7, pp. 861–870, 1991.

[19] A. R. Conn, N. I. M. Gould, and P. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 433–460, 1988. See also same journal, 26:764–767, 1989.

[20] A. R. Conn, N. I. M. Gould, and P. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.

[21] A. R. Conn, N. I. M. Gould, and P. L. Toint, LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*, vol. 17 of *Springer Series in Computational Mathematics*. Heidelberg, Berlin, New York: Springer Verlag, 1992.

[22] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 USER'S GUIDE," Tech. Rep. SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA 94305, USA, 1987.

[23] I. Bongartz, A. R. Conn, N. I. M. Gould, and P. L. Toint, "CUTE: Constrained and unconstrained testing environment," *ACM Transactions on Mathematical Software*, vol. 21, no. 1, pp. 123 – 160, 1995.

[24] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Quassemzadeh, T. R. Scott, W. Warth, and J. Werner, "Algorithms for ASTAP - a network analysis program," *IEEE Transactions on Circuit Theory*, vol. CT-20, pp. 628–634, 1973.

[25] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic Circuit and System Simulation Methods.* New York: McGraw-Hill, 1995.

[26] T. V. Nguyen, P. Feldmann, S. W. Director, and R. A. Rohrer, "SPECS simulation validation with efficient transient sensitivity computation," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 252–255, 1989.

[27] P. Feldmann, T. V. Nguyen, S. W. Director, and R. A. Rohrer, "Sensitivity computation in piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 2, pp. 171–183, 1991.

[28] T. V. Nguyen, "Transient sensitivity computation and application," Technical Report CMUCAD-91-40, Carnegie Mellon University, Pittsburgh, USA, 1991.

[29] R. K. Brayton and S. W. Director, "Computation of delay time sensitivities for use in time domain optimization," *IEEE Transactions on Circuits and Systems*, vol. CAS-22, no. 12, pp. 910–920, 1975.

[30] M. R. Hestenes, "Multiplier and gradient methods," *Journal of Optimization Theory and Applications*, vol. 4, pp. 303–320, 1969.

[31] M. J. D. Powell, "A method for nonlinear constraints in minimization problems," in *Optimization* (R. Fletcher, ed.), London and New York: Academic Press, 1969.

[32] B. D. H. Tellegen, "A general network theorem, with applications," *Philips Research Reports*, vol. 7, pp. 259–269, 1952.

[33] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[34] J. J. Moré, "Recent developments in algorithms and software for trust region methods," in *Mathematical Programming: The State of the Art* (A. Bachem, M. Grötschel, and B. Korte, eds.), pp. 258–287, Berlin: Springer Verlag, 1983.

[35] A. R. Conn, N. I. M. Gould, and P. L. Toint, "On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds," in *Proceedings of the 14th Biennial Numerical Analysis Conference Dundee 1991* (D. Griffiths and G. Watson, eds.), pp. 49–68, Longmans, 1992.

[36] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization.* London and New York: Academic Press, 1981.

[37] R. B. Schnabel and E. Eskow, "A new modified Cholesky factorization," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, pp. 1136–1158, 1991.

[38] A. R. Conn, L. N. Vicente, and C. Visweswariah, "Two-step algorithms for nonlinear optimization with structured applications," Research Report (in preparation), IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA, 1997.

[39] D. P. Bertsekas, *Constrained Optimization and Lagrange Multipliers Methods.* London: Academic Press, 1982.

[40] Cadence Design Systems Inc., "Design entry: Composer users' guide 4.3," 1994.

[41] M. Rubin, "View: A user tailorable interface for circuit design graphics," Research Report TR-19.90629, IBM, 1990.

[42] A. R. Conn and N. I. M. Gould, "An exact penalty function for semi-infinite programming," *Mathematical Programming*, vol. 37, no. 1, pp. 19–40, 1987.

[43] M. D. Matson and L. A. Glasser, "Macromodeling and optimization of digital MOS VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, no. 4, pp. 659–678, 1986.

Reply to Comments of Reviewer 1

1. In order not to increase the length of the paper too much we have refrained from including a detailed description of sensitivity computation and Lagrangian optimization methods, but refer the reader to the references instead.

2. The definitions for "process corner" and "adjoint Lagrangian group" has been added.

Reply to Comments of Reviewer 2

1. We believe that the reduction in the number of variables is not artificial, but is a result of the design and layout process where large circuits are not built from scratch, but rather common building blocks are replicated in many areas of the same design. Furthermore, we take the point of view that the designer has a general grasp of how the circuit functions and is in a position to choose the independent parameters. This is especially true when there are important layout considerations that the designer takes into account. Therefore, when the problem is presented to Jiffytune, we assume that the independent parameters have been chosen and variables have been grouped. The Cadence interface in Jiffytune makes it easy for the designer to specify which parameters are dependent and independent.

2. The actual numbers used for displaying Figure 5 are shown in Table 3, so detailed information is available to the reader in Table 3.