

Power Grid Verification Using Node and Branch Dominance *

Nahi Abdul Ghani
 ECE Department
 University of Toronto
 Toronto, Ontario, Canada
 nahi@eecg.utoronto.ca

Farid N. Najm
 ECE Department
 University of Toronto
 Toronto, Ontario, Canada
 f.najm@utoronto.ca

ABSTRACT

The verification of power grids in modern integrated circuits must start early in the design process when adjustments can be most easily incorporated. This work describes a vectorless verification technique that deals with circuit uncertainty in the framework of current constraints. In such a framework, grid verification becomes a question of computing the worst-case voltage drops which, in turn, entails the solution of as many linear programs (LPs) as there are nodes. First, we extend grid verification to also check for the worst-case branch currents. We show that this would require as many LPs as there are branches. Second, we propose a starkly different approach to *reduce* the number of LPs in the verification problem. We achieve this by examining *dominance* relations among node voltage drops and among branch currents. This allows us to replace a group of LPs by one conservative and tight LP. Results show a dramatic reduction in the number of LPs thus making vectorless grid verification in the framework of current constraints practical and scalable.

Categories and Subject Descriptors

B.7.2 [Hardware]: Integrated Circuits—*Design Aids*

General Terms

Performance, Algorithms, Verification

Keywords

Power grid, voltage drop, dominance

1. INTRODUCTION

The rising demand for low-voltage modern integrated circuits (ICs) has made efficient analysis of power grids a critical task. A key challenge is that such analysis must start early in the design phase, and often prior to the completion of the underlying logic circuitry. This is the time when grid modifications can be most easily incorporated. Therefore, in an effort to mitigate circuit uncertainty, design teams typically either rely on their previous designs or employ over-designed grids. Such measures, however, do not obviate the risk that some parts of the power grid may remain prone to violations due to un-anticipated circuit switching activities. There is a need, then, for an *early* verification approach that identifies power grid violations while accounting for circuit uncertainty.

We adopt the framework of partial current specifications in the form of *current constraints* [1]. These constraints specify a feasible space of current waveforms. In such a framework, grid verification requires the computation of the worst-case voltage drops

*This work was supported by the Semiconductor Research Corporation (SRC) (www.src.org).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2011, June 5-10, 2011, San Diego, California, USA
 Copyright 2011 ACM 978-1-4503-0636-2/11/06 ...\$10.00.

at all grid nodes and for all feasible input currents [2, 3, 4]. This is not an easy problem, and previous works tackled it by solving a linear program (LP) for every node on the grid, a method that becomes too expensive for large grids. In [3], the authors took advantage of grid locality to generate a reduced-size LP for every node while ensuring a user-specified over-estimation margin (in volts) on the solution of the exact LPs. The authors of [4], on the other hand, proposed a dual approach to solve each of the LP problems. The approach eliminates some of the decision variables and constraints thus achieving speed-ups over the exact solution. In an effort to speed-up the verification process, all of these formulations attempted to reduce the size of every LP but, still, *full* grid verification requires the application of as many LPs as there are nodes, a requirement that represents a significant part of the verification cost.

In this work, we first extend grid verification to incorporate the verification of branch currents, which allows one to check for potential failures in wires arising from high current transients. We derive tight and conservative bounds on the worst-case branch currents and show that these bounds, like their voltage drop counterparts, require as many LPs as there are branches. Second, unlike prior art, we leverage grid locality to capture what we call *dominance* relations among node voltage drops and among branch currents. As we will see, this novel technique allows us to efficiently *replace* a group of LPs by one *conservative* and *tight* LP. Experimental results in section 5 show that the reduction in the number of LPs, and consequently in the verification CPU time, is dramatic while the errors introduced are minimal.

2. BACKGROUND

2.1 The Power Grid Model

Consider an *RC* model of the power grid, where each branch is represented by a resistor and where there exists a capacitor from every node to ground. In a power grid, some nodes have ideal current sources (to ground) representing the currents drawn by the circuits tied to the grid at these nodes, while other nodes may be connected to ideal voltage sources representing the external voltage supply (V_{dd}). Let the power grid consist of n non- V_{dd} nodes. We can then write the RC model for the power grid as [5]:

$$C\dot{v}(t) + Gv(t) = i_s(t) \quad (1)$$

where $v(t)$ is the $n \times 1$ vector of time varying voltage drops (difference between V_{dd} and node voltages) and $i_s(t)$ is the $n \times 1$ non-negative vector of all current sources connected to the grid. We assume that $\forall k = 1, \dots, n$, $i_{s,k}(t)$ is well-defined, so that nodes with no current sources attached have $i_{s,k}(t) = 0$. C is the $n \times n$ diagonal capacitance matrix, since all capacitors were assumed to be node-to-ground. G is the $n \times n$ conductance matrix, which is known to be a diagonally-dominant symmetric positive definite \mathcal{M} -matrix (so that $G^{-1} \geq 0$). Using the finite difference approximation $(\dot{v}(t) \cong \frac{v(t) - v(t - \Delta t)}{\Delta t})$, a discrete-time version of (1) can be written as:

$$Av(t) = \frac{C}{\Delta t}v(t - \Delta t) + i_s(t) \quad (2)$$

where $A = (G + \frac{C}{\Delta t})$. It can also be shown that A is a symmetric positive definite \mathcal{M} -matrix, so that $A^{-1} \geq 0$.

As we mentioned earlier, as part of grid verification, in this work we investigate not only the worst-case voltage drops, but also the worst-case branch currents. Notice that (1) does not explicitly describe the branch currents. Let m be the number of branches in the grid and let $i_{b,l}(t)$ represent the branch currents, where $l = 1 \dots m$, and let $i_b(t)$ be the vector of all branch currents. The necessary equations were derived in [6] and are simply stated here. Relating all branch currents to the voltage drops across them we get:

$$i_b(t) = -R^{-1}M^T v(t) \quad (3)$$

where R is an $m \times m$ diagonal matrix of the branch resistance values, and M is an $n \times m$ incidence matrix whose elements are ± 1 or 0, as in [6]. The term ± 1 occurs in location m_{kl} of the matrix when node k is connected to the l th branch, else a 0 occurs. The sign of the non-zero terms depends on the node under consideration. If the current assignment is away from the node, then the sign is positive, else it is negative.

2.2 Constraint-Based Framework

As in [1], we deal with source current uncertainties within the framework of linear constraints. In particular, we distinguish *local constraints* and *global constraints*. A local constraint represents a bound on the peak value of the current drawn by a current source. A current source can represent a single logic gate or cell, but more typically should represent a larger block. Local constraints can be expressed as:

$$0 \leq i_s(t) \leq i_L, \quad \forall t \geq 0 \quad (4)$$

where i_L is the vector of peak values that the current sources can draw. Global constraints are upper bound constraints on the sums of certain subsets of current sources. They reflect design expertise, and they are meant to reduce pessimism in grid verification by incorporating engineering judgment about circuit specification or functionality at an early stage. Assuming we have a total of κ global constraints, they can be expressed in matrix form as:

$$0 \leq U i_s(t) \leq i_G, \quad \forall t \geq 0 \quad (5)$$

where U is a $\kappa \times n$ matrix that consists only of 0s and 1s which indicate which current sources are present in each global constraint. Together, the local and global constraints define a *feasible space* of currents, which we denote by \mathcal{F} , so that $i_s(t) \in \mathcal{F}$ if and only if it satisfies (4) and (5). Later in the paper, we will define algorithms that operate on a vector $i_s(t)$ and which are applicable at any value of time t . In that context, we will use the shorthand $i_s \in \mathcal{F}$ to denote the fact that $i_s(t)$ is feasible, for any t .

3. PROBLEM DEFINITION

In the constraint-based framework, grid verification becomes a question of computing the worst-case voltage drops at all grid nodes, for all feasible currents. This proves to be a difficult task as it requires solving as many linear programs (LPs) as there are nodes [1, 2, 3].

In this work, we extend grid verification to include the verification of branch currents on the grid. We believe this will be very useful in order to check for catastrophic failures in wires resulting from high current, or to check for electromigration. First, we derive tight and conservative bounds on the worst-case branch currents on the grid. These bounds also turn out to require as many LPs as there are branches. Second, contrary to prior art, we follow a starkly different approach to reduce the number of LPs in computing the worst-case voltage drops and the worst-case branch currents. We propose a new technique that examines relations among node voltage drops and among branch currents. This allows us to replace a group of LPs by *one conservative and tight* LP. We refer to such relations as *dominance* relationships. However, before we proceed, we need to provide some background to motivate our contribution.

For grid verification under the constraint-based framework, and at every point in time t , the source current vector $i_s(t)$ must be feasible, *i.e.*, we must have $i_s(t) \in \mathcal{F}$ and under this condition, we are interested in the worst-case behavior attained (separately) by each component of $v(t)$ and $i_b(t)$. In [3], the authors showed that the worst-case voltage drops on the grid can be written as:

$$v_{max}(t) = \lim_{p \rightarrow \infty} \sum_{k=0}^{p-1} \text{emax}_{\forall i_s \in \mathcal{F}} \left[\left(A^{-1} \frac{C}{\Delta t} \right)^k A^{-1} i_s \right] \quad (6)$$

where the “emax(.)” notation denotes *element-wise* maximization¹. Following a similar analysis and using (3) and (6), one can easily show that the general solution to the worst-case branch currents is:

$$i_{b,opt}(t) = \lim_{p \rightarrow \infty} \sum_{k=0}^{p-1} \text{eopt}_{\forall i_s \in \mathcal{F}} \left[-R^{-1} M^T \left(A^{-1} \frac{C}{\Delta t} \right)^k A^{-1} i_s \right] \quad (7)$$

where $i_{b,opt}(t)$ is a $2m \times 1$ vector, and where the “eopt(.)” notation denotes element-wise optimization. It is decomposed into emax(.) and emin(.) operators where each component $i_{b,l,opt}(t)$, $\forall l = 1, \dots, m$ can be found using the emax operator and each component $i_{b,m+l,opt}(t)$, $\forall l = 1, \dots, m$ can be found using an emin operator (element-wise minimization). Both a maximization and a minimization are required in (7) because branch currents can flow in either direction. In fact, the maximum current magnitude in any branch l is given by:

$$i_{b,l,max}(t) = \max \{ |i_{b,l,opt}(t)|, |i_{b,m+l,opt}(t)| \} \quad (8)$$

Unfortunately, both (6) and (7) are of theoretical interest only. They cannot be directly computed, as they stand, because they have to be evaluated for a large number of time steps until convergence and because the element-wise optimizations require a matrix inverse A^{-1} and *linear programs* (LPs) that are proportional to the number of nodes or branches in the grid which for modern designs is in the millions. Also, they include matrix-matrix multiplications at every time step.

In previous works, various approaches to simplify (6) have been proposed. In [2], the authors proposed an upper bound on the exact worst-case voltage drop vector. As $t \rightarrow \infty$, the bound was found to be:

$$v_{ub} = \left(I + G^{-1} \frac{C}{\Delta t} \right) v_a, \quad v_a = \text{emax}_{\forall i_s \in \mathcal{F}} A^{-1} i_s \quad (9)$$

where I is the identity matrix. The result in (9) has a big run-time advantage over (6) as it requires only a single emax(.) operation. However, the formulation involves the inverse of a large matrix A , which is expensive to compute. In [3], the authors proposed an efficient approach to compute tight conservative bounds on v_a based on a sparse approximation of the matrix inverse. Since $G^{-1} \geq 0$ and $C \geq 0$, this in turn guaranteed a conservative approximation of v_{ub} . Still, the method required as many LPs as there are nodes.

In the rest of the paper, we first derive an upper bound on the maximum worst-case branch current and a lower bound on the minimum worst-case branch current in every branch of the grid. The bounds will be analogous to (9) in the sense that they will initially require a matrix inverse, *one* application of the eopt(.) operator, and *one* application of the emax(.) operator. To speed up the matrix inverse, we adapt the approach in [3] to compute a sparse approximation of the matrix inverse while ensuring user-controlled conservative approximations on the upper and lower bounds. Even in this case, the number of LPs will be proportional to the size of the grid and to the number of grid branches. Therefore, instead of performing all the optimizations, we explore *dominance* relationships among node voltage drops and among branch currents. As we will see, this allows for a drastic reduction in the number of LPs performed for both verification problems, the voltage verification problem in (9) and the branch current verification derived in this work, while maintaining the bounds’ conservatism as $t \rightarrow \infty$.

4. PROPOSED SOLUTION

4.1 Upper and Lower Bounds

Let $D = -R^{-1}M^T$ and $B = A^{-1}C/\Delta t$, then using (2) and (3), we can write:

$$i_b(t) = DBv(t - \Delta t) + DA^{-1}i_s(t) \quad (10)$$

Let P be the matrix whose non-zero entries are the non-negative entries of DB , and let N be the element-wise non-positive matrix $N = DB - P$. Since $v(t) \geq 0$ [5], and by using the fact that P is non-negative and N is non-positive, we can write the following:

¹Finding each element of the “emax(.)” requires solving an LP, as defined in [3].

$$Nv(t - \Delta t) \leq DBv(t - \Delta t) \leq Pv(t - \Delta t) \quad (11)$$

We also have for all $i_s(t) \in \mathcal{F}$:

$$\min_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \leq DA^{-1}i_s(t) \leq \max_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s)$$

Adding the above two equations leads to:

$$\begin{aligned} Nv(t - \Delta t) + \min_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) &\leq i_b(t) \\ &\leq Pv(t - \Delta t) + \max_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \end{aligned} \quad (12)$$

Recall from [5] that the worst-case voltage drop was attained by starting the grid in a zero state at time 0, then allowing for any feasible combination of current waveforms for all $t > 0$. In that formulation, the true worst-case voltage drop is attained as $t \rightarrow \infty$, and is upper bounded by v_{ub} in (9). Using a similar approach for currents, and enforcing $t \rightarrow \infty$ in (12), gives:

$$\begin{aligned} N \lim_{t \rightarrow \infty} v(t - \Delta t) + \min_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) &\leq \lim_{t \rightarrow \infty} i_b(t) \\ &\leq P \lim_{t \rightarrow \infty} v(t - \Delta t) + \max_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \end{aligned} \quad (13)$$

Using (9), one can say that $\lim_{t \rightarrow \infty} v(t - \Delta t) \leq v_{ub}$. Since P is a non-negative matrix, and N is a non-positive matrix, we get: $Nv_{ub} \leq N \lim_{t \rightarrow \infty} v(t - \Delta t)$ and $P \lim_{t \rightarrow \infty} v(t - \Delta t) \leq Pv_{ub}$. Combining these results with (13) yields:

$$Nv_{ub} + \min_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \leq \lim_{t \rightarrow \infty} i_b(t) \leq Pv_{ub} + \max_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \quad (14)$$

This leads to the desired upper and lower bounds on the maximum and minimum branch currents, respectively, as:

$$\begin{aligned} i_{b,max} &\leq Pv_{ub} + \max_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \\ i_{b,min} &\geq Nv_{ub} + \min_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \end{aligned} \quad (15)$$

The result in (15) has a run-time advantage over (7) as it requires one $\max(\cdot)$ operation of size n (required for v_{ub} computation) and then one $\max(\cdot)$ and one $\min(\cdot)$ operation each of size m :

$$x^* = \max_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s), \quad y^* = \min_{\forall i_s \in \mathcal{F}} (DA^{-1}i_s) \quad (16)$$

Once these are computed, the evaluation of (15) is relatively cheap. It involves two matrix-vector multiplications and two vector additions.

4.2 Efficient Bounds Computation

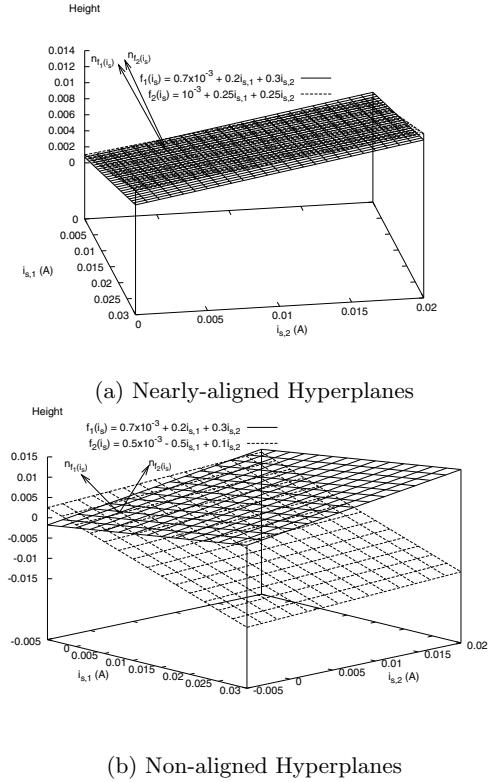
Like (9), computing (16) involves the inverse of the large matrix A . The authors in [3] proposed an efficient approach to compute a tight conservative approximation of v_a in (9) by using a sparse approximation of the matrix inverse. We show that using that same approach would guarantee an efficient computation of DA^{-1} which will be ultimately used to get a conservative approximation to x^* and y^* . It will also be shown that using this sparse approximation ensures a *user-specified* error margin (in amperes). We proceed by briefly reviewing the technique employed in [3] and then, in the following section, we describe how we adapted it to suit our needs.

4.2.1 Voltage Bounds Review

Let Q be a variable $n \times n$ matrix and let the vector q_k denote the k th column of Q . Let e_k be the $n \times 1$ vector consisting of all zeros, except for its k th component which is 1. Then, consider the $n \times 1$ vector, called the k th residual, defined by:

$$r_k = Aq_k - e_k, \quad \forall k = 1, \dots, n$$

It is clear that, if $Q = A^{-1}$, then $r_k = 0, \forall k$. In general, the norm of the residual $\|r_k\|_2$ is always positive and becomes zero only when $Q = A^{-1}$. Therefore, the norms of the residuals, for all k , provide a measure of how far Q is from being equal to the



(a) Nearly-aligned Hyperplanes

(b) Non-aligned Hyperplanes

Figure 1: Two Hyperplanes

desired matrix inverse A^{-1} . Let $Q^* \triangleq A^{-1}$, so that when Q takes the value Q^* , then $\|r_k\|_2 = 0, \forall k$. In [3], the authors employed a sparse approximate inverse technique (SPAI) [7] to find an upper bound \bar{v}_a on v_a . Given some user-defined tolerance η , SPAI computes an approximation \hat{Q} to the actual inverse matrix Q^* . SPAI starts with an initial $Q = I$, and tries to refine its columns by minimizing $\|r_k\|_2, \forall k = 1, \dots, n$ subject to η . Throughout the process, SPAI strives to maintain sparsity and reduce fill-ins. In order for SPAI to compute a *conservative* approximation to the $\max(\cdot)$ in (9), the authors modified SPAI so that it stops when:

$$\|\hat{q}_k - q_k^*\|_\infty \leq \epsilon_k, \quad \forall k = 1, \dots, n \quad (17)$$

where $\|\cdot\|_\infty$ is the infinity norm and $\epsilon_k > 0$ is an internal error tolerance that is derived from another user-specified error-tolerance, δ , on voltages, based on:

$$\epsilon_k = \frac{\delta}{u_k^T i_L + u^T i_L}, \quad \forall k = 1, \dots, n \quad (18)$$

where u is an $n \times 1$ vector of all 1s, and u_k is another $n \times 1$ vector with 1s only in locations corresponding to non-zero entries in \hat{q}_k , and otherwise 0. When (17) is met, the result $\|\bar{v}_a - v_a\|_\infty \leq \delta$ is achieved, where \bar{v}_a is computed as:

$$\bar{v}_a = \max_{\forall i_s \in \mathcal{F}} (\hat{Q}i_s) + (u^T i_L)\epsilon \quad (19)$$

and where u is the same vector of all ones, and $\epsilon = [\epsilon_1 \ \epsilon_2 \ \dots \ \epsilon_n]^T$.

4.2.2 Current Bounds

Suppose that SPAI terminates with its approximation matrix \hat{Q} , using the same stopping criterion introduced in [3], and assume we ran our $\max(\cdot)$ and $\min(\cdot)$ operators in (16) using \hat{Q} instead of Q^* , to get:

$$\hat{x} = \max_{\forall i_s \in \mathcal{F}} (D\hat{Q}i_s), \quad \hat{y} = \min_{\forall i_s \in \mathcal{F}} (D\hat{Q}i_s) \quad (20)$$

Theorem 1. Let ζ be a user-specified error tolerance on branch currents (in amperes), and suppose we run SPAI with an error tolerance of:

$$\delta = \frac{\zeta/4}{\|R^{-1}\|_\infty} \quad (21)$$

Then, an upper bound, denoted by \bar{x}_l , on x_l^* , and a lower bound, denoted by \bar{y}_l , on y_l^* , $\forall l = 1, \dots, m$, are obtained as follows:

$$\bar{x}_l = \hat{x}_l + \alpha_l (\epsilon_j u^T i_L + \epsilon_i u_i^T i_L) \quad (22)$$

$$\bar{y}_l = \hat{y}_l - \alpha_l (\epsilon_i u^T i_L + \epsilon_j u_j^T i_L) \quad (23)$$

where α_l is the conductance of branch l , and where u is the same vector of all 1s introduced above, and u_i and u_j , also introduced earlier, are vectors with 1s only in locations corresponding to non-zero entries in \hat{q}_i and \hat{q}_j , respectively. Such bounds achieve the result that:

$$\|\bar{x} - x^*\|_\infty \leq \zeta, \quad \|y^* - \bar{y}\|_\infty \leq \zeta$$

The proof of this result is omitted for lack of space.

4.3 Using Dominance Relationships

We have seen that finding (an upper bound v_{ub} on) the worst-case voltage drop, based on (9) and (19), requires one $\text{emax}(\cdot)$ operation. As well, finding (bounds on) the worst-case maximum and minimum branch currents, based on (15), (16), (20), (22), and (23), requires an additional $\text{emax}(\cdot)$ and $\text{emin}(\cdot)$ operation. These emax and emin computations represent a significant part of the cost of grid verification, because each one of them requires running a large number of LPs, one for each element of the emax or emin solution vector. Basically, one has to run an LP for every grid node to find its worst-case voltage drop, and another two LPs for each branch to find its worst-case maximum and minimum currents. However, intuitively, but also due to grid topology and grid locality [8], we know that grid node voltages are not totally independent, nor are branch currents. Therefore, to benefit from this, so that we hopefully do not have to run LPs for *every* grid node and branch, we have looked for ways to capture what we call *dominance* relations among node voltage drops and among branch currents, as we will see below. To verify the grid, it will be sufficient to verify only the nodes and branches whose hyperplanes are dominant.

4.3.1 Hyperplane Dominance and Alignment

With an appropriately defined vector h and matrix H , it is clear that (19), (22), and (23) can be expressed as:

$$\begin{aligned} \bar{v}_a &= h + \underset{\forall i_s \in \mathcal{F}}{\text{emax}}(H i_s) = \underset{\forall i_s \in \mathcal{F}}{\text{emax}}(h + H i_s) \\ \bar{x} &= h + \underset{\forall i_s \in \mathcal{F}}{\text{emax}}(H i_s) = \underset{\forall i_s \in \mathcal{F}}{\text{emax}}(h + H i_s) \\ \bar{y} &= -h + \underset{\forall i_s \in \mathcal{F}}{\text{emin}}(H i_s) = \underset{\forall i_s \in \mathcal{F}}{\text{emin}}(-h + H i_s) \end{aligned} \quad (24)$$

where, obviously, h and H would be defined differently in each of the above three equations. The point is, the *form* of the equations is the same. In every case, we want to separately maximize or minimize every row of the vector given by $\pm h + H i_s$. Generically, we can represent any such row (i) as the affine expression:

$$f_i(i_s) = h_{i0} + \left(\sum_{j=1}^w h_{ij} \times i_{s,j} \right), \quad \forall i = 1, \dots, z \quad (25)$$

where w , z , h_{ij} , and the offset h_{i0} depends on whether we are solving for \bar{v}_a , \bar{x} , or \bar{y} . From (25), we see that each objective function can be viewed as a *hyperplane* in the current space. One dimension is a generic dimension (generic, in the sense that it can represent the value of any objective function under consideration), while all other dimensions are the currents at all the current sources. Examples of such hyperplanes are given in Fig. 1, where the vertical dimension denotes the generic dimension that gives $f_i(\cdot)$ and the other two horizontal dimensions denote two source currents. We define a hyperplane to be *dominant* over another if it is larger than (or equal to) it at all feasible points i_s , when finding $\text{emax}(\cdot)$, while in the case $\text{emin}(\cdot)$, dominance denotes a less than (or equal to) relationship. In this work, we will look for either existing or artificial new hyperplanes that dominate over others. To be viable, the method has to meet three important

criteria. First, the resulting maximum (minimum) hyperplane should act as a ceiling (floor) to all given hyperplanes at every point in the current space. Second, the over/under-estimation introduced should be minimal. Third, the method should be computationally efficient.

It is easy to replace a group of hyperplanes by a new hyperplane that is larger (smaller) (at all points in the current domain) than the initial group of hyperplanes. Since the values of the current sources can only be positive, then a maximum hyperplane is readily obtained via an element-wise maximum of the coefficients and the offsets. As an example, consider the hyperplanes in Fig. 1(a). Applying the element-wise maximum on the coefficients and the offsets, we obtain a hyperplane whose equation is $10^{-3} + 0.25i_{s,1} + 0.3i_{s,2}$. Clearly, this hyperplane is a maximum on both $f_1(i_s)$ and $f_2(i_s)$ at every point in the current space. Similarly, a minimum hyperplane is obtained via an element-wise minimum of the coefficients and the offsets. However, for the over/under-estimation to be minimal, it is necessary that we consider hyperplanes that satisfy the following two key criteria. First, the hyperplanes should have approximately equal offsets. Second, they should be *nearly aligned* with each other. The first criterion is satisfied by construction; by a close examination of (18), (19), and (21-23), and by the definitions of vectors u , u_i , u_j , and u_k , one can easily see that $0 \leq h_{i0} \leq \delta$, $\forall i$, in the case of \bar{v}_a , and $0 \leq |h_{i0}| \leq \zeta/2$, $\forall i$, in the case of \bar{x} and \bar{y} . As we will see in section 5, the user-controlled error tolerances, δ and ζ , are chosen to be very small values so that we achieve minimal errors. As a result, and in what follows, we will focus on the hyperplane alignment.

To better visualize this, consider the example in Fig. 1, which shows two sample hyperplanes with approximately equal offsets. In the first scenario, the hyperplanes are nearly aligned in space as shown in Fig. 1(a). In this case, finding the maximum/minimum hyperplane in the above mentioned fashion will result in minimal over/under-estimation. Fig. 1(b), on the other hand, shows two non-aligned hyperplanes in space. It is clear, in this case, that the maximum hyperplane will result in excessive over-estimation on $f_2(i_s)$. Similarly, the minimum hyperplane will result in excessive under-estimation on $f_1(i_s)$. From this, it is clear that misalignment of hyperplanes should be avoided to maintain low over/under-estimation.

The measure of alignment between any two hyperplanes, p_1 and p_2 , can be captured by either of the two supplementary angles between the normal vectors to the hyperplanes [9]. These angles are computed through the normalized dot product of the normal vectors. In fact, the cosine of the acute angle θ between the respective normal vectors \vec{o}_1 and \vec{o}_2 can be expressed in terms of the absolute value of their normalized dot product as follows [9]:

$$\cos(\theta) = \left| \frac{\vec{o}_1 \cdot \vec{o}_2}{\|\vec{o}_1\| \times \|\vec{o}_2\|} \right| \quad \text{where } 0 \leq \theta \leq \frac{\pi}{2}$$

Therefore, a large (close to 1) absolute value of the normalized dot product corresponds to a small θ , meaning that p_1 and p_2 are almost exactly aligned in space. On the other hand, a small absolute value of the normalized dot product implies a large θ which means that the hyperplanes are not aligned. For the hyperplanes in Fig. 1(a), the absolute value of the normalized dot product of their normal vectors is 0.9978, while for the hyperplanes in Fig. 1(b), the absolute value of the normalized dot product is 0.7794.

4.3.2 Overview of Approach

Given all this, we will in the next two sub-sections describe the first two key phases of our approach. In the first phase (section 4.3.3), we will find the normalized dot products among pairs of hyperplanes, using a very efficient sparse-matrix based approach. With this in hand, we then in the second phase (section 4.3.4) proceed to eliminate many hyperplanes depending on their alignment, in two ways: 1) if two hyperplanes are well-aligned, we will remove them both and replace them by a new hyperplane that dominates them both, and 2) if a number of hyperplanes are well-aligned with the current subspace (the subspace corresponding to $f_i(i_s) = 0$), then we will remove them all and replace them by a new hyperplane that dominates them all. We would then be left with a much smaller number of hy-

Algorithm 1 compute_maximum_hyperplanes

Input: S, U, λ
Output: \mathcal{H}_{max}

- 1: $\mathcal{H}_{max} = \emptyset$
- 2: **for all** $(u_{eg}, e \neq g)$ such that $|u_{eg}| \geq \lambda$ **do**
- 3: **if** $(f_e(i_s) \notin S \text{ and } f_g(i_s) \notin S)$ **then**
- 4: $h_m(i_s) = \text{element_wise_max}(f_e(i_s), f_g(i_s))$
- 5: $\mathcal{H}_{max} = \mathcal{H}_{max} \cup \{h_m(i_s)\}$
- 6: $S = S - \{f_e(i_s), f_g(i_s)\}$
- 7: **else if** $(f_e(i_s) \notin S \text{ and } f_g(i_s) \in S)$ **then**
- 8: $\exists h_m(i_s) \in \mathcal{H}_{max}: h_m(i_s)$ is element-wise maximum on $f_e(i_s)$
- 9: $h_n(i_s) = \text{element_wise_max}(h_m(i_s), f_g(i_s))$
- 10: $\mathcal{H}_{max} = \mathcal{H}_{max} \cup \{h_n(i_s)\}$
- 11: $S = S - \{f_g(i_s)\}$
- 12: **else if** $(f_e(i_s) \in S \text{ and } f_g(i_s) \notin S)$ **then**
- 13: $\exists h_m(i_s) \in \mathcal{H}_{max}: h_m(i_s)$ is element-wise maximum on $f_g(i_s)$
- 14: $h_n(i_s) = \text{element_wise_max}(h_m(i_s), f_e(i_s))$
- 15: $\mathcal{H}_{max} = \mathcal{H}_{max} \cup \{h_n(i_s)\}$
- 16: $S = S - \{f_e(i_s)\}$
- 17: **else if** $(f_e(i_s) \notin S \text{ and } f_g(i_s) \notin S)$ **then**
- 18: //Do Nothing
- 19: **end if**
- 20: **end for**
- 21: Let $h_p(i_s)$ represent the source current sub-space
- 22: **for all** $(u_{ee} \text{ such that } f_e(i_s) \in S \text{ and } |u_{ee}| \geq \lambda)$ **do**
- 23: $h_p(i_s) = \text{element_wise_max}(f_e(i_s), h_p(i_s))$
- 24: $S = S - \{f_e(i_s)\}$
- 25: **end for**
- 26: $\mathcal{H}_{max} = \mathcal{H}_{max} \cup \{h_p(i_s)\}$
- 27: $\mathcal{H}_{max} = \mathcal{H}_{max} \cup S$

perplanes, which we call *the dominant hyperplanes*. Finally, in a third phase of our approach, and going back to equation (24), any rows in h and H corresponding to hyperplanes that are dominated by others are replaced by the coefficients of those dominant hyperplanes. Every dominant hyperplane then leads to the application of only one LP as part of the relevant $\text{emax}(\cdot)/\text{emin}(\cdot)$ operation, and all the rows of the $\text{emax}(\cdot)/\text{emin}(\cdot)$ dominated by that hyperplane will have been thereby automatically “solved”. The results will provide conservative tight bounds for the hyperplanes (rows) that were removed.

4.3.3 Computing Normalized Dot Products

We first present a way to compute the normalized dot product between any two hyperplanes in (25). We follow that with the computation of the normalized dot product between any hyperplane in (25) and the hyperplane representing the current sub-space. Finally, we discuss the efficiency of our computation method.

Alignment between any two hyperplanes in (25): Consider any two hyperplanes $f_e(i_s)$ and $f_g(i_s)$ in (25) with $e \neq g$. According to [9], their normal vectors can be expressed using the coefficient of the generic dimension and the coefficients of $i_{s,j}$ in (25) as follows:

$$\begin{aligned}\vec{n}_{f_e} &= (1, -h_{e1}, -h_{e2}, \dots, -h_{ew})^T \\ \vec{n}_{f_g} &= (1, -h_{g1}, -h_{g2}, \dots, -h_{gw})^T\end{aligned}$$

The normalized dot product of these vectors is:

$$\frac{\vec{n}_{f_e} \cdot \vec{n}_{f_g}}{\|\vec{n}_{f_e}\| \times \|\vec{n}_{f_g}\|} = \frac{1 + \sum_{k=1}^w h_{ek} h_{gk}}{\sqrt{1 + \sum_{k=1}^w h_{ek}^2} \times \sqrt{1 + \sum_{k=1}^w h_{gk}^2}} \quad (26)$$

Note that $\sum_{k=1}^w h_{ek} h_{gk}$ is simply the entry of HH^T with row index e and column index g , where H is the $z \times w$ matrix consisting of the h_{ij} entries in (25). Similarly, $\sum_{k=1}^w h_{ek}^2$ is the e th diagonal entry, and $\sum_{k=1}^w h_{gk}^2$ is the g th diagonal entry. Therefore, an efficient way to compute the normalized dot product is to perform a *sparse* matrix multiplication of H and its transpose. Since the result is symmetric, we need only to compute the upper triangular part or the lower triangular part. In this work, we compute U , the upper triangular part of HH^T . Then, and in order to compute the normalized dot product $\forall e, g, e \neq g$, we traverse the matrix U , and for every *non-zero* entry u_{eg} , we update it, by replacing it by:

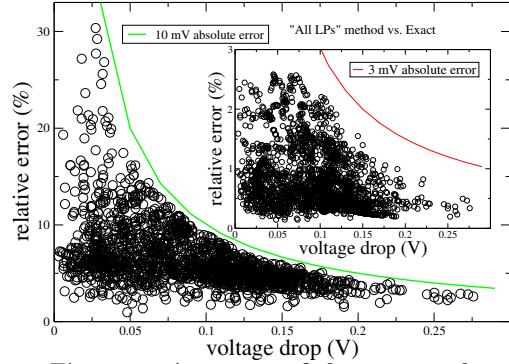


Figure 2: Accuracy of the proposed approach

$$\frac{1 + u_{eg}}{\sqrt{1 + u_{ee}} \times \sqrt{1 + u_{gg}}}, \quad \text{when } e \neq g \quad (27)$$

which gives us the desired normalized dot product (26).

Alignment between a hyperplane in (25) and the current sub-space: From Fig. 1, one can see that the current sub-space is represented as $f_i(i_s) = 0$. Consequently, a normal vector to the current sub-space can be expressed as $(1, 0, 0, \dots, 0)^T$. Then, in order to compute the normalized dot product between $f_e(i_s)$ and the current sub-space, we update the diagonal entries u_{ee} for all $e = 1, \dots, z$, by replacing it by:

$$\frac{1}{\sqrt{1 + u_{ee}}} \quad (28)$$

Complexity: This method of computing the normalized dot products is extremely efficient as it will result in far less entries than $(z^2 + z)/2$ (the result if we were to compute the normalized dot product for every two objective functions $f_e(i_s)$ and $f_g(i_s)$ for all $e = 1, \dots, z$ and $g = 1, \dots, z$, and between every objective function $f_e(i_s)$ and the current sub-space). The is due to the sparsity of H which, in turn, arises from the sparsity of \hat{Q} [3]. Another consequence of the sparsity of H is that, every objective function $f_i(\cdot)$ would be dependent on only a few current sources, and there may be many pairs of objective functions that are dependent on mutually exclusive sets of current sources. For any two such objective functions there is no point in examining their alignment in space. The sparse matrix multiplication automatically detects this, and the corresponding entry in U will be zero.

4.3.4 The Dominance Algorithm

After U is computed, and once its entries are updated using (27) and (28), we proceed with the computation of the dominant hyperplanes. The algorithm takes as input the full set of hyperplanes, defined in (25). Let S denote this set. The other inputs include the matrix U , and some user-defined tolerance λ , where λ is a non-zero positive number less than, but close to, 1. The algorithm returns as output the set of dominant hyperplanes. Note that in the case of \bar{v}_a and \bar{x} , maximum hyperplanes are constructed. Let \mathcal{H}_{max} denote the set of such hyperplanes. In the case of \bar{y} , however, minimum hyperplanes are constructed. Let \mathcal{H}_{min} represent the set of minimum hyperplanes. Initially the size of both \mathcal{H}_{max} and \mathcal{H}_{min} is zero. Algorithm 1, `compute_maximum_hyperplanes`, computes \mathcal{H}_{max} . It employs a sub-routine, `element_wise_max`, that computes a dominating hyperplane on any two hyperplanes via an element-wise maximum of the coefficients and the offsets. Computing \mathcal{H}_{min} can be also done using Algorithm 1, but with the sub-routine `element_wise_max` replaced by another subroutine that generates a dominating hyperplane via an element-wise minimum of the coefficients and the offsets.

Algorithm 1 starts by examining the off-diagonal entries u_{eg} , $e \neq g$, in U . Recall that such entries now correspond to the normalized dot product between the normal vectors to the e th and g th hyperplanes. If $|u_{eg}| \geq \lambda$, then the corresponding hyperplanes are nearly aligned in space and finding a maximum hyperplane on them would result in minimal over-estimation. Depending on whether any of the hyperplanes was considered previously, four cases arise and are considered in steps 3, 7, 12, and 17 of the algorithm.

Table 1: Speed and accuracy Using Node Dominance

Power Grid		Dominance Method		Comparison to “all LPs” Method		Cost of “all LPs” Method (CPU time)
Name	Nodes	CPU time	% Node LP Elimination	Max. voltage drop (mV)	Absolute error (mV)	
G1	2,213	12.16 sec.	45.30	276.73	8.69	20.81 sec.
G2	4,822	27.73 sec.	46.53	196.95	6.62	45.84 sec.
G3	8,413	52.47 sec	45.39	181.35	6.33	80.76 sec.
G4	24,668	166.16 sec.	47.25	133.84	4.48	248.31 sec.
G5	50,444	344.27 sec.	46.75	117.21	2.61	520.2 sec.
G6	113,304	657.48 sec.	46.78	134.17	3.01	1039.50 sec.

Table 2: Speed and accuracy Using Branch Dominance

Power Grid		Dominance Method			Comparison to “all LPs” Method		Cost of “all LPs” Method (CPU time)	
Name	Nodes	Branches	CPU time	% Node LP Elimination	% Branch LP Elimination	Max. branch current (mA)	Absolute error (mA)	
G1	2,213	3,269	31.97 sec.	45.45	44.08	180.75	9.95	57.76 sec.
G2	4,822	7,155	73.06 sec.	47.1	44.29	141.59	7.56	119.54 sec.
G3	8,413	12,512	149.57 sec.	45.63	33.25	133.46	2.54	215.46 sec.
G4	24,668	36,808	511.38 sec.	47.66	21.08	124.63	1.064	656.85 sec.
G5	50,444	75,505	1035.22 sec.	46.98	18.47	76.39	1.13	1689.64 sec.
G6	113,304	169,160	1829.55 sec.	44.94	25.33	131.04	3.97	3266.96 sec.

Next, we look at the diagonal entries of U corresponding to the remaining hyperplanes. For all those whose absolute values are larger than λ , the corresponding hyperplanes are nearly aligned with the current sub-space. As shown in steps 21-26, such hyperplanes will be combined, using `element_wise_max`, to produce one maximum hyperplane to be added to \mathcal{H}_{max} . In step 27, any hyperplane that is not considered yet is added as is to \mathcal{H}_{max} .

5. EXPERIMENTAL RESULTS

The algorithm in section 4.3.4, along with the computations from section 4.3.3, have been implemented in C++. To solve the required linear programs, we used Mosek [10]. We carried out experiments on a set of randomly-generated power grids, using a 2.6 GHz Linux machine with 24 GB of RAM. Moreover, all experiments were performed on grids with up to ten global constraints, 1.1V supply voltage, $\delta = 1mV$, $\zeta = 1mA$, and $\lambda = 0.95$.

To study the accuracy of our dominance algorithm, Fig. 2 shows a scatter plot of the resulting relative errors, in percent, versus the exact maximum worst-case voltage drops on a 2,213-node grid. Our dominance algorithm was applied to the voltage drop verification problem in (19). This was then followed by an application of (9). The exact maximum worst-case voltage drops were computed using the exact verification approach in (6). In this exact approach, the worst-case drop at every node is obtained by solving a set of LPs at consecutive time steps until convergence. The figure also shows the curve corresponding to an absolute error of 10 mV where a point on the curve represents a node voltage drop that is over-estimated by exactly 10 mV. A point with an over-estimation greater than 10 mV would lie in a region above the curve. From the resulting scatter plot, it is clear that the absolute errors, arising from the dominance algorithm, are very small. This implies that we are able to drastically reduce the number of LPs ($\sim 45\%$ elimination in this case) without sacrificing much accuracy. We also show another scatter plot embedded in Fig. 2. It compares the accuracy of solving all the LPs in (19), followed by (9), versus the exact method in (6). As seen from both plots, the absolute errors arising from using the dominance algorithm are only $\sim 7 mV$ larger than when solving all LPs.

Table 1 shows the speed and accuracy of our dominance algorithm when applied to the verification problem corresponding to node voltage drops. Again, we compare our results with those resulting from solving all LPs in (19). In order to compute the voltage drop bounds at infinity, both scenarios are followed up by an application of (9). As an accuracy measure, we report the maximum voltage drop on the grid and the corresponding absolute error. The number of grid nodes is reported in column 2, and the percentage of LPs eliminated is reported in column 4. We also give the run-times of both approaches. The CPU time for solving all the LPs in (19) is given in column 7, whereas the CPU time to solve the LPs arising from the dominance algorithm is reported in column 3. Note that this CPU time also includes the time for the computations in section 4.3.3, and the time for applying the dominance algorithm in section 4.3.4. For all the grids under study, results show that our proposed approach eliminates almost half of the LPs while incurring relatively small absolute errors, never

exceeding 10 mV. This elimination of LPs translates into considerable reduction in verification run-times ranging from around 42% (G1) to around 33% (G4).

Similarly, table 2 shows the performance of the dominance algorithm when applied to the branch current verification problem (22-23). Note that the dominance algorithm is also performed on (19) because v_{ub} is part of the branch current verification. We compare our results with those arising from solving all these LPs. To compute the maximum and minimum branch currents at infinity, both scenarios are followed up by an application of (15). For all the grids under study, results show that our proposed approach eliminates almost half of the node LPs while the percentage of eliminated branch current LPs varies from around 44% (G1) to around 18% (G5). The absolute errors incurred, on the largest current magnitude, are also small, and the reduction in run-time due to the node and branch LP elimination ranges from around 45% (G1) to around 22% (G4).

6. CONCLUSION

We extend an early grid verification approach to incorporate the verification of branch currents. To that end, we derive tight and conservative bounds on the worst-case branch currents and show that these bounds, like their voltage drop counterparts, require as many LPs as there are branches. We then propose a novel technique that examines dominance relations among node voltage drops and among branch currents. It allows us to replace a group of LPs by one conservative and tight LP. Results show a dramatic reduction in the number of LPs while incurring minimal errors.

7. REFERENCES

- [1] D. Kouroussis and F. N. Najm. A static pattern-independent technique for power grid voltage integrity verification. In *ACM/IEEE DAC*, pages 99–104, Anaheim, CA, Jun. 2-6 2003.
- [2] I. A. Ferzli, F. N. Najm, and L. Kruze. A geometric approach for early power grid verification using current constraints. In *ACM/IEEE ICCAD*, pages 40–47, San Jose, CA, Nov. 5-8 2007.
- [3] N. H. Abdul Ghani and F. N. Najm. Fast vectorless power grid verification using an approximate inverse technique. In *IEEE/ACM DAC*, San Francisco, CA, Jul. 26-31 2009.
- [4] X. Xiong and J. Wang. An efficient dual algorithm for vectorless power grid verification under linear current constraints. In *IEEE/ACM DAC*, pages 837–842, Anaheim, CA, Jun. 13-18 2010.
- [5] M. Nizam, F. N. Najm, and A. Devgan. Power grid voltage integrity verification. In *ACM/IEEE ISLPED*, pages 239–244, San Diego, CA, Aug. 8-10 2005.
- [6] T. R. Bashkow. The A matrix, new network description. *IRE Transactions on Circuit Theory*, 4(3), September 1957.
- [7] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, May 1997.
- [8] E. Chiprout. Fast flip-chip power grid analysis via locality and grid shells. In *ACM/IEEE ICCAD*, pages 485–488, San Jose, CA, Nov. 7-11 2004.
- [9] E. S. Smith, M. Salkover, and H. K. Justice. *Analytic geometry*. J. Wiley, New York, 1954.
- [10] The mosek optimization software. <http://www.mosek.com/>.