A geometric approach for early power grid verification using current constraints*

Imad A. Ferzli
Department of ECE
University of Toronto
Toronto, Ontario, Canada
ferzli@eecg.utoronto.ca

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

Lars Kruse
Magma Design Automation
Eindhoven, The Netherlands
lars@magma-da.com

ABSTRACT

The verification of power grids in modern integrated circuits must start at design time, where circuit information is unknown but could be specified or inferred from design or architectural considerations. This work builds on previously proposed techniques to deal with circuit uncertainty in the framework of linear current constraints, but proposes a cost-controlled solution, by following a geometric approach, and transforming a problem that requires as many linear programs as there are power grid nodes, to another involving a user-limited number of solutions of one linear system.

1. INTRODUCTION

Power grid verification must start at design-time. For one thing, power routing resources must be committed in the early stages, often prior to completion of circuit design itself. However, existing power grid verification techniques and commercial tools assume a fully designed circuit and can only be useful after placement and routing, or for final signoff, As a result, design groups typically start with what they view, based on previous experience, as over-designed grids, without a way to factor in circuit uncertainty to reduce the extent of over-design. Relying on previous design experience alone cannot, however, preclude that some sections of the power grid may remain susceptible to voltage violations, such as localized dynamic voltage drop due to switching current activity. Therefore, there is a need to systematically *prototype the grid* early in the design flow, i.e., estimate its worst-case voltage drops, while accounting for circuit uncertainty.

A critical aspect of circuit uncertainty, that which is the focus of this work, is the imprecise characterization of circuit currents. To capture this uncertainty, we adopt the framework of current constraints [1], which specify a feasible space in which currents can vary during circuit operation. Grid prototyping and verification become a question of computing the maximum voltage drops everywhere on the grid, for all feasible currents. This is not an easy problem, and previous work [1] tackled it by solving a linear program (LP) for every node on the grid at consecutive time steps. Finding the maximum voltage drop everywhere on the grid would therefore require as many LPs as there are nodes, which becomes too expensive for large grids. The present work follows a starkly different method, by exploiting the particularities of the power grid at design time and the geometry of the current feasibility space to derive an efficient solution. A preliminary version of this work appeared in [2].

2. PROBLEM FORMULATION

2.1 Constraint-Based Framework

Consider a power grid with n nodes, m of which have a current source tied to them. Assuming, without loss of generality, that nodes with a current source are numbered $1, \ldots, m$, we can write the RC-model for the power grid as:

$$\mathbf{C}\dot{\mathbf{v}}(t) + \mathbf{G}\mathbf{v}(t) = \begin{bmatrix} \mathbf{i}(t) \\ \mathbf{0_{(n-m)}} \end{bmatrix} = \begin{bmatrix} \mathbf{I_{m\times m}} \\ \mathbf{0} \end{bmatrix} \mathbf{i}(t) = \mathbf{H}\mathbf{i}(t), \quad (1)$$

The system equation (1) can be formulated [1] so that n is specifically the number of nodes that do not correspond to V_{dd} sites, and $\mathbf{v}(t)$ is the n-vector of time varying voltage drops (difference between V_{dd} and node voltages). $\mathbf{i}(t)$ is the m-vector of current loads, $\mathbf{0_{(n-m)}}$ is the (n-m)-zero-vector, and $\mathbf{H}$ is an $(n \times m)$ matrix whose top $(m \times m)$ block is the m-dimensional identity matrix $\mathbf{I_{m\times m}}$, and bottom block is 0. Assuming all capacitance is node-to-ground, then $\mathbf{C}$ is the $n \times n$ diagonal capacitance matrix, and $\mathbf{G}$ is the $n \times n$ conductance matrix and is known to be a symmetric positive definite M-matrix [3]. For the purposes of power grid verification and optimization, DC analysis plays an important role in some practical design flows. The system equation for a DC grid model is a direct analogue of (1), and is given by $\mathbf{G}\mathbf{v} = \mathbf{H}\mathbf{i}$. We draw this parallel because the techniques proposed in this paper apply equally to both DC and RC models.

As stated earlier, we deal with circuit current uncertainties within the framework of linear constraints [1]. In particular, we distinguish local constraints and global constraints. A local constraint represents a bound on the peak value of current drawn by a current source over time. For example, if current source $i_j(t)$ does not draw more than l_j, we write $0 \le i_j(t) \le l_j$. Writing a similar inequality for every current source leads to the following component-wise inequality:

$$\mathbf{0} \le \mathbf{i}(t) \le \mathbf{I_L} \quad (2)$$

such that the j^{th} component of $\mathbf{I_L}$ is l_j. Global constraints are upper bound constraints on the sum of a certain subset of current sources. They are user-supplied, reflect design expertise, and are meant to reduce pessimism in voltage drop estimation by incorporating engineering judgment about circuit specification or functionality at an early stage. Suppose there are c global constraints, $\mathcal{G}_0, \ldots, \mathcal{G}_{c-1}$, then we write the global constraints as:

$$\sum_{j=0}^{m-1} u_{ij} i_j(t) \le g_i, \qquad i = 0, \ldots, c-1, \quad (3)$$

where u_{ij} is an indicator variable assuming the value of 1 if i_j is included in $\mathcal{G}_i$, and 0 otherwise.

Together, (2) and (3) define a feasibility region for circuit currents, denoted by $\mathcal{I}_{\mathcal{F}}$. We seek the vector of maximum node voltage drops on the grid, at any point in time, as currents vary inside $\mathcal{I}_{\mathcal{F}}$. The exact solution to this problem is prohibitive and involves solving one LP over $\mathbb{R}^n$, for every node on the grid. This work suggests an alternative strategy which efficiently leads to a conservative estimate of this vector.

2.2 Vector of Upper Bounds

As a matter of notation, we first state that all inequalities in this paper, when applied on vectors and matrices, and all min/max operators applied on vectors, are component-wise. Let

*This research was supported by Intel Corp., by Altera Corp., and by the SRC (www.src.org).

$\mathbf{A} = \mathbf{G} + \mathbf{C}/\Delta t$, which can be shown to be a positive definite M-matrix [1]. We start by time-discretizing (1):

$$\mathbf{v}(t) = \mathbf{A^{-1}Hi}(t) + \mathbf{A^{-1}}(\mathbf{C}/\Delta t)\mathbf{v}(t - \Delta t). \quad (4)$$

$\mathbf{A}$ being an M-matrix, $\mathbf{A^{-1}} \geq 0$, therefore:

$$\mathbf{v}(t) \leq \max_{\mathbf{i}(t)\in\mathcal{I}_\mathcal{F}} \mathbf{A^{-1}Hi}(t) + \mathbf{A^{-1}}(\mathbf{C}/\Delta t)\mathbf{v_{ub}}(t - \Delta t), \quad (5)$$

where $\mathbf{v_{ub}}(t - \Delta t)$ represents a vector of upper bounds on the maximum voltage drop at time step $t - \Delta t$, and $\max \mathbf{A^{-1}Hi}(t)$ is a vector whose j^{th} component is the maximum of $\mathbf{a_j Hi}(t)$, as $\mathbf{i}(t)$ varies in $\mathcal{I}_\mathcal{F}$, where $\mathbf{a_j}$ is the j^{th} row of $\mathbf{A^{-1}}$.

Although the RC model features dynamic currents and voltages, the description of local and global constraints is static, i.e., $\mathbf{I_L}$ in (2) and g_i in (3) do not depend on time and $\mathcal{I}_\mathcal{F}$ is the same for each time step. Therefore, $\max \mathbf{A^{-1}Hi}(t)$, $\mathbf{i}(t) \in \mathcal{I}_\mathcal{F}$, is independent of t, and we will denote it by $\mathbf{V_a}$. We have from (5):

$$\mathbf{v_{ub}}(t) = \mathbf{V_a} + \mathbf{A^{-1}}(\mathbf{C}/\Delta t)\mathbf{v_{ub}}(t - \Delta t) \quad (6)$$

is an upper bound on $\mathbf{v}(t)$. Denote by $\mathbf{v}(0)$ the vector of voltage drops on the grid at $t = 0$, i.e., the grid's initial condition. This leads to $\mathbf{v_{ub}}(0) = \mathbf{v}(0)$ and $\mathbf{v_{ub}}(\Delta t) = \mathbf{V_a} + \mathbf{A^{-1}}(\mathbf{C}/\Delta t)\mathbf{v}(0)$. Since $\mathbf{A^{-1}} \geq \mathbf{0}$ and $\mathbf{B} = \mathbf{A^{-1}C}/\Delta t \geq \mathbf{0}$, writing (6) at time steps $\Delta t, \ldots, k\Delta t$ yields:

$$\mathbf{v_{ub}}(k\Delta t) \quad = \quad (\mathbf{I} + \mathbf{B} + \ldots + \mathbf{B^{k-1}})\mathbf{V_a} + \mathbf{B^k}\mathbf{v}(0), \quad (7)$$

where $\mathbf{I}$ denotes the identity matrix. The convergence of $\mathbf{v_{ub}}$, as $k \to \infty$, depends on the convergence of a) the matrix series $\sum_{k=0}^{\infty} \mathbf{B^k}$ and b) the matrix sequence $\mathbf{B^k}$, as $k \to \infty$.

The series $\sum_{k=0}^{\infty} \mathbf{B^k}$ is known to converge [3] if and only if $\rho(\mathbf{B}) < 1$, where $\rho(\mathbf{B})$ is the magnitude of the largest eigenvalue of $\mathbf{B}$, under which condition the series limit is $(\mathbf{I} - \mathbf{B})^{-1}$. The condition that $\rho(\mathbf{B}) < 1$ is also necessary and sufficient for the convergence of the sequence $\mathbf{B^k}$ to 0 [3]. We will now prove that $\rho(\mathbf{B}) < 1$, and find a limiting value for $\mathbf{v_{ub}}(k\Delta t)$, independent of the initial condition $\mathbf{v}(0)$, by making use of the fact that $\mathbf{B} \geq 0$ and of the following theorem [3]:

Theorem 1. *Let $\mathbf{B}$ be a nonnegative matrix. Then $\rho(\mathbf{B}) < 1$ if and only if $\mathbf{I} - \mathbf{B}$ is nonsingular and $(\mathbf{I} - \mathbf{B})^{-1}$ is nonnegative.*

Our key convergence result is captured in the following claim:

Corollary 1. *$\mathbf{v_{ub}}(k\Delta t)$ converges to $\mathbf{V_u} = (\mathbf{I} + \mathbf{G^{-1}C}/\Delta t)\mathbf{V_a}$, as $k \to \infty$, for all $\Delta t > 0$.*

PROOF. $\mathbf{B} = \mathbf{A}^{-1}(\mathbf{A} - \mathbf{G}) = \mathbf{I} - \mathbf{A^{-1}G}$, $\mathbf{I} - \mathbf{B} = \mathbf{A^{-1}G}$, and $(\mathbf{I} - \mathbf{B})^{-1} = \mathbf{G^{-1}A} = \mathbf{I} + \mathbf{G^{-1}C}/\Delta t$. $\mathbf{A}$ and $\mathbf{G}$ being positive definite, $\mathbf{det}(\mathbf{A^{-1}G}) = \mathbf{det}(\mathbf{G})/\mathbf{det}(\mathbf{A}) \neq 0$ so $\mathbf{I} - \mathbf{B}$ is nonsingular ($\mathbf{det}(.)$ denotes the determinant of a matrix). Since $\mathbf{G}$ is an M-matrix, $\mathbf{G^{-1}} \geq 0$. Given that $\mathbf{C} \geq 0$, we immediately have that $(\mathbf{I} - \mathbf{B})^{-1} \geq \mathbf{0}$. Therefore, by theorem 1, $\rho(\mathbf{B}) < 1$. This implies that $\mathbf{B^k}$ converges to 0 and that $\sum_{k=0}^{\infty} \mathbf{B^k}$ converges to $(\mathbf{I}-\mathbf{B})^{-1} = \mathbf{I}+\mathbf{G^{-1}C}/\Delta t$, as $k \to \infty$, yielding that $\mathbf{v_{ub}}(k\Delta t)$ converges to $(\mathbf{I} + \mathbf{G^{-1}C}/\Delta t)\mathbf{V_a}$. □

We note that the computation of this limit is easy, given $\mathbf{V_a}$: scale the j^{th} component of $\mathbf{V_a}$ by $c_j/\Delta t$ (c_j is the j^{th} component of $\mathbf{C}$), yielding $\mathbf{V_s} = \mathbf{CV_a}/\Delta t$, then solve for $\mathbf{V_b}$ such that $\mathbf{GV_b} = \mathbf{V_s}$ (one standard linear system solve). The upper bound is the sum $\mathbf{V_u} = \mathbf{V_a} + \mathbf{V_b}$. Therefore, $\mathbf{V_u}$ can be easily deduced from $\mathbf{V_a}$, given a standard *LU* factorization of $\mathbf{G}$.

The problem of finding a vector of upper bounds on the voltage drop maxima given an RC grid model is thus reduced to finding $\mathbf{V_a} = \max \mathbf{A^{-1}Hi}$ over all $\mathbf{i} \in \mathcal{I}_\mathcal{F}$. The problem is similar when a DC grid model is used: we must find $\max \mathbf{G^{-1}Hi}$ over all $\mathbf{i} \in \mathcal{I}_\mathcal{F}$, with the exception that the DC solution yields the exact vector of voltage drop maxima, not a bound thereon.

We digress to mention that, when transient analysis is used, $\mathbf{V_a}$ depends on the choice of the time step Δt: since $\mathbf{V_a}$ is the upper bound on the voltage drops at $t = \Delta t$, the smaller the time step, the smaller $\mathbf{V_a}$. In fact, although the upper bound converges for any $\Delta t > 0$, choosing too small a time step leads to overestimating the maximum voltage drops in a way that could be avoided. To see why, observe that in the above derivation of the maximum voltage drop upper bound, we are implicitly assuming that currents may change arbitrarily in their feasibility region $\mathcal{I}_\mathcal{F}$ within Δt time. This would allow, for example, that currents switch from 0 to their maximum possible values, i.e., their local constraints, or the other way around, within a single time step. This is clearly not the case for arbitrarily small Δt. With this in mind, we note that problems with a larger time step are subsets of problems with a smaller time step, by virtue of the fact that maximizations need to be performed at a greater number of time points when smaller time steps are used. This results in the observation that the smaller Δt, the larger $\mathbf{V_u}$, componentwise. Therefore, the upper bound resulting from unrealistically small Δt is too pessimistic an estimate. On the other hand, the time step needs to be small enough to capture the transition times on the grid voltages. Therefore, design expertise needs to guide the choice of Δt by striking a balance between the dynamics of the current loads and voltage responses on the grid, in order to avoid pessimism in the computation of the voltage drop upper bound. An alternative, which could obliviate this question altogether, is the use of dynamic current constraints, i.e., making $\mathcal{I}_\mathcal{F}$ time-dependent. This may be more difficult in practice, both from the user's standpoint (supplying dynamic constraints) as well as the tool's (dealing with them). However, dynamic constraints may afford greater accuracy in voltage drop estimation. Either way, dealing with dynamic current constraints is part of our ongoing research, and this papers considers only static constraints.

Going back to the estimation of $\mathbf{V_u}$, observe that, for a given Δt, if $\mathbf{V_{a,ub}} \geq \mathbf{V_a}$, then $(\mathbf{I} + \mathbf{G^{-1}C}/\Delta t)\mathbf{V_{a,ub}} \geq \mathbf{V_u}$, i.e., a conservative estimate of $\mathbf{V_a}$ leads to a conservative estimate of $\mathbf{V_u}$. In what follows, we propose an approach to efficiently compute a conservative estimate of $\mathbf{V_a} = \max \mathbf{A^{-1}Hi}$, where $\mathbf{A}$ is understood to mean $\mathbf{G}$ in case of DC analysis.

3. COMPUTATIONAL STRATEGY

3.1 Finding $\mathbf{V_a}$ at The Vertices of $\mathcal{I}_\mathcal{F}$

Every local and global current constraint corresponds to a hyperplane (more precisely, to a half-space of a hyperplane) in $\mathbb{R}^m$, and the feasibility region $\mathcal{I}_\mathcal{F}$ is a convex polytope [4] formed by the intersection of all these half-spaces. A global constraint $\mathcal{G}_i$, for example, of the form $\sum u_{ij} i_j \leq g_i$, as in (3), defines a hyperplane $\mathcal{H}_{\mathcal{G}_i} : \sum u_{ij} i_j = g_i$, and two half-spaces $\mathcal{H}^-_{\mathcal{G}_i} : \sum u_{ij} i_j \leq g_i$ (belongs to $\mathcal{G}_i$), and $\mathcal{H}^+_{\mathcal{G}_i} : \sum u_{ij} i_j > g_i$ (outside of $\mathcal{G}_i$). Note that, from (2), two hyperplanes are associated with each local constraint: one for the lower bound of 0, and one for the upper bound. If we have c global constraints, then $\mathcal{I}_\mathcal{F}$ consists of the intersection of $(2m+c)$ half-spaces. Of interest are the vertices of $\mathcal{I}_\mathcal{F}$. A detailed discussion of hyperplanes, polytopes, and vertices is beyond the scope of this paper, and we refer the reader to [4] for a comprehensive discussion. It suffices to say that a vertex in $\mathcal{I}_\mathcal{F}$ is formed by the intersection of any m of the $(2m + c)$ constraint hyperplanes, provided this intersection exists and belongs to $\mathcal{I}_\mathcal{F}$.

Notice that the local constraints alone form a hypercube in $\mathbb{R}^m$, with 2^m vertices at $[0/l_0, 0/l_1, \ldots, 0/l_{m-1}]^T$. The j^{th} local constraint, therefore, can be either "turned ON" in a vertex if its corresponding entry is l_j, or "turned OFF" if 0. Denoting the local constraint hypercube by $\mathcal{K}$, we can write $\mathcal{I}_\mathcal{F}$ as

$$\mathcal{I}_\mathcal{F} = \mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_0} \cap \mathcal{H}^-_{\mathcal{G}_1} \ldots \cap \mathcal{H}^-_{\mathcal{G}_{c-1}}. \quad (8)$$

Recall from section 2.2 that the j^{th} component of $\mathbf{V_a}$ is the maximum of $\mathbf{a_j^T Hi}$, $\mathbf{i} \in \mathcal{I}_\mathcal{F}$, therefore, the solution of a linear program. From the theory of linear programming [4], we know that the maximum occurs at a vertex of $\mathcal{I}_\mathcal{F}$. This is true for every component of $\mathbf{V_a}$. As a result, the j^{th} component of $\mathbf{V_a}$ is the maximum value attained by the j^{th} component of $\mathbf{A^{-1}Hi}$ over all the vertices of the polytope $\mathcal{I}_\mathcal{F}$. We express this as:

$$\mathbf{V_a} = \max_{\mathbf{i}\in\mathcal{V}(\mathcal{I}_\mathcal{F})} \mathbf{A^{-1}Hi} \quad (9)$$

where $\mathcal{V}(\mathcal{I}_\mathcal{F})$ denotes the set of vertices of $\mathcal{I}_\mathcal{F}$. Therefore, $\mathbf{V_a}$ can be computed in two steps: 1) Solve $\mathbf{A^{-1}Hi}$ at all vertices of $\mathcal{I}_\mathcal{F}$. Let $\mathcal{S}$ be the set of solution vectors 2) The j^{th} component of $\mathbf{V_a}$ is the maximum value of the j^{th} component of all vectors in $\mathcal{S}$.

Alone, however, this procedure is insufficient, since the number of vertices in $\mathcal{I}_\mathcal{F}$ may be too large [4]. In what follows, we refer to solving $\mathbf{A^{-1}Hi}$ at a particular vertex $\mathbf{i}$ as *visiting* vertex $\mathbf{i}$.

Our overall strategy, described in the next three sub-sections, is to enlarge the feasibility region $\mathcal{I}_\mathcal{F}$ in such a way as to yield a user-limited number of vertices that need to be visited in order to compute a conservative estimate of $\mathbf{V_a}$ while limiting the computational cost. The first such enlargement of $\mathcal{I}_\mathcal{F}$ is by forming the intersection of $\mathcal{K}$ with each global constraint separately.

3.2 Applying One Global Constraint at A Time

Noting that $\mathcal{I}_\mathcal{F} \subset \mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$, $\forall i$, we can write

$$\mathbf{V_a} \leq \min_{i=0,\ldots,c-1} \left(\max_{\mathbf{i} \in \mathcal{V}(\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i})} \mathbf{A^{-1}Hi} \right) \tag{10}$$

where the minimum is component-wise. By "decoupling" the global constraints in this way, we have reduced our problem to maximizing $\mathbf{A^{-1}Hi}$ over the polytope $\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$, formed by the intersection of one hyperplane with a hypercube. Since $\mathcal{I}_\mathcal{F} \subset \mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$, maximizing a function over $\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$ overestimates the maximum of that function over $\mathcal{I}_\mathcal{F}$, with the advantage that the vertices of $\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$ are fewer and easier to compute than those of $\mathcal{I}_\mathcal{F}$. The accuracy tradeoff is to the extent of overlap among global constraints: if no two global constraints share the same current source, then (10) is an equality. In practice, this is not a problem at the design planning stage since currents represent high-level blocks and global constraints are user-supplied, one could formulate them with little or no overlap, but this is not a requirement or limitation of our work - global constraints *may* overlap.

3.3 Vertex Dominance

Not all vertices of $\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$ need to be visited: this section shows that it is enough to only visit the vertices which belong to $\mathcal{H}_{\mathcal{G}_i}$, or a subset thereof. Because $\mathbf{A^{-1}} \geq 0$, we can write

$$\mathbf{i_1} \geq \mathbf{i_2} \Rightarrow \mathbf{A^{-1}Hi_1} \geq \mathbf{A^{-1}Hi_2}. \tag{11}$$

In other words, increasing any coordinate (component) of $\mathbf{i}$ is guaranteed to not decrease any component of $\mathbf{A^{-1}Hi}$.

We say that $\mathbf{i_1}$ dominates $\mathbf{i_2}$ if $\mathbf{i_1} \geq \mathbf{i_2}$. In this case, and given (11), we can state that, when looking for $\max(\mathbf{A^{-1}Hi})$, it is sufficient to visit $\mathbf{i_1}$ (the dominating vertex), and there is no need to visit $\mathbf{i_2}$. Similarly, we define dominance between sets of current vertices: If $\mathcal{V}_1$ and $\mathcal{V}_2$ are two sets of current vertices, we say that $\mathcal{V}_1$ dominates $\mathcal{V}_2$ if every vertex in $\mathcal{V}_2$ is dominated by at least one vertex in $\mathcal{V}_1$.

Let $\dim(\mathcal{G}_i)$ be the number of current sources included in $\mathcal{G}_i$, i.e., the number of indicator variables u_{ij} that are 1 in the expression of $\mathcal{G}_i$, given in (3). We refer to $\dim(\mathcal{G}_i)$ as the dimension of $\mathcal{G}_i$. To a vertex $\mathbf{i} \in \mathbb{R}^m$, we associate a rank $r(\mathbf{i}) = \sum_{j=0}^{m-1} \mathbf{i}(j)$, where $\mathbf{i}(j)$ is the j^{th} coordinate of $\mathbf{i}$, i.e., $\mathbf{i} = [\mathbf{i}(0) \ldots \mathbf{i}(m-1)]^T$.

We state without proof the following proposition. For a proof, the reader is referred to [2]:

Proposition 1. *If $dim(\mathcal{G}_i) = m$, then*

$$\max_{\mathbf{i} \in \mathcal{V}(\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i})} \mathbf{A^{-1}Hi} = \max_{\mathbf{i} \in \mathcal{V}(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})} \mathbf{A^{-1}Hi} \tag{12}$$

If $\dim(\mathcal{G}_i) = m\prime < m$, then

$$\max_{\mathbf{i} \in \mathcal{V}(\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i})} \mathbf{A^{-1}Hi} = \max_{\mathbf{i} \in \mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})} \mathbf{A^{-1}Hi} \tag{13}$$

When $\dim(\mathcal{G}_i) = m$, all vertices of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}$ share the same rank g_i, so none dominates another (e.g. $\mathbf{i_4}$ and $\mathbf{i_5}$ in Fig. 1(a)). Equation (12) therefore implies that all vertices lying at the intersection of $\mathcal{H}_{\mathcal{G}_i}$ and $\mathcal{K}$ must be visited.

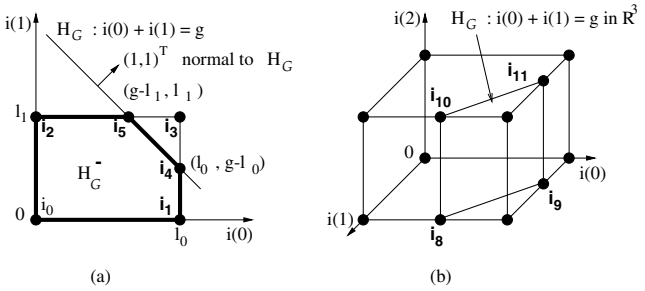

Figure 1: (a) In $\mathbb{R}^2$, $\mathcal{K}$ is a rectangle and $\mathcal{H}_\mathcal{G}$ an intersecting straight line. $\mathcal{V}(\mathcal{K} \cap \mathcal{H}_\mathcal{G}) = \{\mathbf{i_4}, \mathbf{i_5}\}$ dominates $\mathcal{V}(\mathcal{K} \cap \mathcal{H}^-_\mathcal{G})$. (b) In $\mathbb{R}^3$, $\dim(\mathcal{G}) = 2$, $\mathcal{V}^0(\mathcal{K} \cap \mathcal{H}_\mathcal{G}) = \{\mathbf{i_8}, \mathbf{i_9}\}$, $\mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_\mathcal{G}) = \{\mathbf{i_{10}}, \mathbf{i_{11}}\}$. $\mathcal{V}^1(\mathcal{K} \cap \mathcal{G})$ dominates $\mathcal{V}(\mathcal{K} \cap \mathcal{H}_\mathcal{G})$.

In case $\dim(\mathcal{G}_i) = m\prime < m$, we distinguish two subsets of $\mathcal{V}(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$: the subset $\mathcal{V}^0(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$, which includes all vertices where all currents not included in $\mathcal{G}_i$ are OFF (e.g., $\mathbf{i_8}$ and $\mathbf{i_9}$ in Fig. 1(b)), and the subset $\mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$, which includes all vertices where all currents not included in $\mathcal{G}_i$ are set to their local constraint values (e.g., $\mathbf{i_{10}}$ and $\mathbf{i_{11}}$ in Fig 1(b)). Equation (13) implies that we only need to visit $\mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}) \subset \mathcal{V}(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$. Notice that vertices in $\mathcal{V}^0(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$ can be thought of as being in $\mathbb{R}^{m\prime}$, since $(m - m\prime)$ of their coordinates are 0.

Computation of the vertices in $\mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$ follows easily from $\mathcal{V}^0(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$: compute the vertices of $\mathcal{V}^0(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$ in $\mathbb{R}^{m\prime}$, then for each such vertex, set every coordinate corresponding to a current source not included in $\mathcal{G}_i$ to its local constraint value, yielding the vertices of $\mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$ in $\mathbb{R}^m$. In the rest of this paper, we may refer to m as the dimension of a global constraint and discuss the computation of vertices of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}$ in $\mathbb{R}^m$, with the understanding that, if $\dim(\mathcal{G}_i) = m\prime < m$, we first compute the vertices of $\mathcal{V}^0(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$ in $\mathbb{R}^{m\prime}$, then deduce those of $\mathcal{V}^1(\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i})$ in $\mathbb{R}^m$.

3.4 Relaxation of The Global Constraint

Based on the above, our problem reduces to visiting the vertices at the intersection of the global constraint hyperplane with the local constraint hypercube. However, the number of these vertices may be too large and their computation expensive [4]. We seek to find a small, user-controlled number of vertices that dominate those of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}$ such that visiting these vertices could be done as quickly as the user can afford, while the estimate of $\max \mathbf{A^{-1}Hi}$ at these vertices would be conservative.

The basic idea is to enlarge the polytope $\mathcal{K} \cap \mathcal{H}^-_{\mathcal{G}_i}$ by suitably shifting the hyperplane $\mathcal{H}_{\mathcal{G}_i}$ in such a way that the shifted hyperplane $\mathcal{H}'_{\mathcal{G}_i}$ satisfies $\mathcal{H}^-_{\mathcal{G}_i} \subset \mathcal{H}'^-_{\mathcal{G}_i}$. Then, the vertices of $\mathcal{K} \cap \mathcal{H}'_{\mathcal{G}_i}$ would dominate those of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}$. We refer to this process as *relaxing the global constraint*. The question becomes how to relax $\mathcal{G}_i$ in such a way that the vertices of $\mathcal{K} \cap \mathcal{H}'_{\mathcal{G}_i}$ are limited in number and easy to compute.

We say that a constraint $\mathcal{G}_i$ (or its hyperplane $\mathcal{H}_{\mathcal{G}_i}$) *leaves out* a vertex $\mathbf{i}$ of $\mathcal{K}$ if $\mathbf{i} \in \mathcal{H}^+_{\mathcal{G}_i}$, otherwise, we say that the vertex is *in the constraint*. The key lies in the fact that we can control the number of vertices of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}$ by controlling the number of vertices of $\mathcal{K}$ left out of the constraint $\mathcal{G}_i$, i.e., the vertices of $\mathcal{K} \cap \mathcal{H}^+_{\mathcal{G}_i}$, based on the following theorem [4]:

Theorem 2. *Consider a polytope $\mathcal{D}$ and a hyperplane $\mathcal{H}$. A point v is a vertex of $\mathcal{D} \cap \mathcal{H}$ if and only if v is either a vertex of $\mathcal{D}$ lying in $\mathcal{H}$ or a point at which an edge $(u\ w)$ of $\mathcal{D}$ intersects $\mathcal{H}$, such that $u \in \mathcal{H}^-$ and $w \in \mathcal{H}^+$.*

The definition of an edge in a general polytope is beyond our scope [4], but we mention that two vertices of $\mathcal{K}$ are neighbors if they differ by exactly one coordinate, and that an edge joins two neighboring vertices. Note that any vertex of $\mathcal{K}$ has exactly m neighboring vertices.

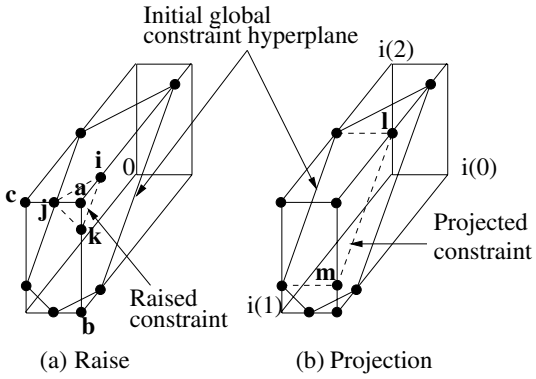

Figure 2: (a) Before the raise of $\mathcal{G}$, $\mathcal{K} \cap \mathcal{H}_\mathcal{G}$ includes five vertices. The raised hyperplane leaves only a out, and its three induced vertices i, j, k, dominate $\mathcal{V}(\mathcal{K} \cap \mathcal{H}_\mathcal{G})$. (b) $\mathcal{G}$ is projected on $\mathbf{i}(1)$ and $\mathbf{i}(2)$. l and m dominate $\mathcal{V}(\mathcal{K} \cap \mathcal{H}_\mathcal{G})$.

Since m edges are incident to every vertex of $\mathcal{K}$, theorem 2 implies that, if $\mathcal{H}^+_{\mathcal{G}_i}$ includes k vertices of $\mathcal{K}$, then the number of vertices of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_i}$ cannot exceed km. We propose two constraint relaxation approaches, based on theorem 2:

1. Raise: If we increase g_i to g^*_i, effectively raising $\mathcal{H}_{\mathcal{G}_i}$ parallel to itself, just enough for the raised hyperplane $\mathcal{H}^*_{\mathcal{G}_i}$ to leave out at most k^* vertices of $\mathcal{K}$, then $\mathcal{H}^-_{\mathcal{G}_i} \subset {\mathcal{H}^*_{\mathcal{G}_i}}^-$ and the number of vertices in $\mathcal{K} \cap \mathcal{H}^*_{\mathcal{G}_i}$ cannot exceed k^*m. See Fig. 2(a). The problem is to find the minimum g^*_i and to compute $\mathcal{V}(\mathcal{K} \cap \mathcal{H}^*_{\mathcal{G}_i})$.

2. Projection: Let $\mathcal{I}_{i,p}$ be a subset of $\{0, 1, \ldots, (m-1)\}$ such that $|\mathcal{I}_{i,p}| = m_p < m$, where $|.|$ denotes the number of elements in a set (recall from section 3.3 that we are assuming $\dim(\mathcal{G}_i) = m$, i.e., that $\mathcal{G}_i$ includes m currents. If $\dim(\mathcal{G}_i) = m\prime < m$, then $\mathcal{I}_{i,p}$ is taken as a subset of the $m\prime$ current source indices included in $\mathcal{G}_i$). If g_i is the value of the global constraint $\mathcal{G}_i$, we define the global constraint $\mathcal{G}_{i,p} : \sum i_j \leq g_i,\ j \in \mathcal{I}_{i,p}$, whose hyperplane $\mathcal{H}_{\mathcal{G}_{i,p}}$ satisfies $\mathcal{H}^-_{\mathcal{G}_i} \subset \mathcal{H}^-_{\mathcal{G}_{i,p}}$. By theorem 2, if $\mathcal{G}_{i,p}$ leaves out no more than k_p vertices of $\mathcal{K}$, then the vertices of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_{i,p}}$ number at most $k_p m_p$. We say that $\mathcal{G}_{i,p}$ is the projection of $\mathcal{G}_i$ on the currents (whose indices are) in $\mathcal{I}_{i,p}$. The problem is to find a suitable subset $\mathcal{I}_{i,p}$ such that $\mathcal{G}_{i,p}$ leaves out at most k_p vertices of $\mathcal{K}$, and to compute the vertices of $\mathcal{K} \cap \mathcal{H}_{\mathcal{G}_{i,p}}$.

A key feature of both relaxations is that the user directly controls the maximum computational cost through the choice of k^* and k_p. Before detailing the process for raising (section 5) and projecting (section 6) a constraint, we lay out the overall solution.

4. TOP-LEVEL ALGORITHM

Recall that visiting a vertex $\mathbf{i}$ means solving the linear system $\mathbf{Av} = \mathbf{Hi}$. Given a factorization of $\mathbf{A}$, this would require a forward/backward substitution at each vertex visited. Although the number of vertices is user-controlled (through the choice of k^* and k_p), it can greatly surpass the number m of current sources. If a forward/backward substitution, which computes the n-vector of power grid voltage drops and is $O(n^2)$, were applied for each vertex visited, visiting all vertices would become too expensive. This, however, is unnecessary and the number of forward/backward substitutions does not need to exceed m: If we choose a basis of vectors in $\mathbb{R}^m$, i.e., m linearly independent m-vectors, we only need a forward/backward substitution at each of the bases (m in total). The solution at any other point in $\mathbb{R}^m$, particularly at a vertex, can be constructed from the solution of the bases, by virtue of system linearity, with the far more efficient *daxpy* operation[3], which, for vectors $\mathbf{x}$ and $\mathbf{y}$ and a scalar α, computes $\alpha\mathbf{x}+\mathbf{y}$, and is $O(n)$. We opted for the natural choice of bases $\mathbf{e_h}$, $h = 0, \ldots, m-1$, for which the h^{th} entry is 1 and all others 0.

Algorithm 1 describes the overall solution. $\mathcal{I}_{i,p}$ is the subset of $\mathcal{I}_i$ retained in the projection of $\mathcal{G}_i$. $\mathcal{V}^{*+}$ and $\mathcal{V}^+_p$ denote the set of current vertices left out of the raised and projected constraint, respectively, and containing at most k^* and k_p vertices each (as per section 3.3, $\mathcal{V}^{*+}$ and $\mathcal{V}^+_p$ need only contain vertices where currents not included in $\mathcal{G}^*_i$ and $\mathcal{G}_{i,p}$ are set to their local constraint values). c is the number of global constraints.

Algorithm 1 returns a vector $\mathbf{V_{ub}}$ of upper bounds on the maximum voltage drop at all the grid nodes.

1: Compute $\mathbf{b_h} = \mathbf{A^{-1}He_h}$, $h = 0, \ldots, m-1$
2: **for** $i = 0, \ldots, c-1$
3: Raise $\mathcal{H}_{\mathcal{G}_i}$ to form $\mathcal{H}^*_{\mathcal{G}_i}$ and $\mathcal{V}^{*+}_i$
4: $\mathbf{V^*_i} = \mathit{COMPUTE\_MAX\_VDROP}\ (\mathcal{H}^*_{\mathcal{G}_i}, g^*_i, \mathcal{I}_i, \mathcal{V}^{*+}_i)$
5: Project $\mathcal{H}_{\mathcal{G}_i}$ on a suitably chosen $\mathcal{I}_{i,p}$ to form $\mathcal{H}_{\mathcal{G}_{i,p}}$ and $\mathcal{V}^+_p$
6: $\mathbf{V_{i,p}} = \mathit{COMPUTE\_MAX\_VDROP}\,(\mathcal{H}_{\mathcal{G}_{i,p}}, g_i, \mathcal{I}_{i,p}, \mathcal{V}^+_p)$
7: $\mathbf{V_i} = \min(\mathbf{V^*_i}, \mathbf{V_{i,p}})$
8: $\mathbf{V_a} \leq \min_{i=0,\ldots,c-1} \mathbf{V_i} = \mathbf{V_{a,ub}}$
9: RC: $\mathbf{V_{ub}} = (\mathbf{I} + \mathbf{G^{-1}C}/\Delta t)\mathbf{V_{a,ub}}$, DC: $\mathbf{V_{ub}} = \mathbf{V_{a,ub}}$

COMPUTE_MAX_VDROP returns the vector of maximum voltage drops induced by a raise (line 4) and a projection (line 6). Note that a raised constraint $\mathcal{G}^*_i$ has a value g^*_i different from g_i, but includes the same currents as $\mathcal{G}_i$, represented in the set $\mathcal{I}_i$. On the other hand, a projected constraint $\mathcal{G}_{i,p}$ has the same value g_i as $\mathcal{G}_i$ but includes a subset $\mathcal{I}_{i,p}$ of the currents in $\mathcal{I}_i$. This explains the second and third arguments of *COMPUTE_MAX_VDROP* on lines 4 and 6. Line 7 follows from the fact that both ${\mathcal{H}^*_{\mathcal{G}_i}}^-$ and $\mathcal{H}^-_{\mathcal{G}_{i,p}}$ include $\mathcal{H}^-_{\mathcal{G}_i}$. Line 8 follows from (10).

Procedure 1 *COMPUTE_MAX_VDROP* ($\mathcal{H}$, g, $\mathcal{I}$, $\mathcal{V}$)

1: $\mathbf{V} = \mathbf{0}$, $\widetilde{\mathbf{V}} = \sum_{j \notin \mathcal{I}} l_j \mathbf{b_j}$, $\tilde{l} = \sum_{j \notin \mathcal{I}} l_j$
2: **for all** vertices $\mathbf{i}^+ \in \mathcal{V}$
3: $\mathbf{V_{i^+}} = \sum_{j \in \mathcal{I}} \mathbf{i}^+(j)\mathbf{b_j}$
4: **for all** $j \in \mathcal{I}$
5: **if** $r(\mathbf{i}^+) - \mathbf{i}^+(j) - \widetilde{\mathbf{V}} \leq g$ **then**
6: $\mathbf{V_{i^*}} = \widetilde{\mathbf{V}} + \mathbf{V_{i^+}} - (r(\mathbf{i}^+) - \tilde{l} - g)\mathbf{b_j}$
7: $\mathbf{V} = \max(\mathbf{V}, \mathbf{V_{i^*}})$
8: **return V**

Procedure 1 describes *COMPUTE_MAX_VDROP*. In this procedure, let $\mathbf{i}^-$ be the j^{th} neighbor of a vertex $\mathbf{i}^+$ (line 2), obtained by changing $\mathbf{i}^+(j)$ to 0 if $\mathbf{i}^+(j) = l_j$. $\mathbf{i}^-$ may be in the global constraint defined by the hyperplane $\mathcal{H}$ *only if* $j \in \mathcal{I}$ (line 4), and is indeed in that global constraint *if* the condition on line 5 is met. In this case, $\exists \mathbf{i}^* \in \mathcal{V}(\mathcal{K} \cap \mathcal{H})$ along the edge from $\mathbf{i}^+$ to $\mathbf{i}^-$, which shares all the coordinates of $\mathbf{i}^+$, except $\mathbf{i}^+(j)$, and is such that the sum of its coordinates corresponding to currents included in $\mathcal{I}$ adds up to g. Therefore, $\mathbf{i}^*(j) = g - (r(\mathbf{i}^+) - \tilde{l} - \mathbf{i}^+(j))$. Letting $\mathbf{V_{i^*}}$ denote the voltage drop at $\mathbf{i}^*$, we have that $\mathbf{V_{i^*}} = \widetilde{\mathbf{V}} + \mathbf{V_{i^+}} - (\mathbf{i}^+(j) - \mathbf{i}^*(j))\mathbf{b_j}$, which reduces to the expression on line 6. Notice that daxpies are performed on lines 1 and 3 to compute $\widetilde{\mathbf{V}}_\mathbf{i}$ and $\mathbf{V_{i^+}}$.

4.1 Complexity

Let us start by computing the complexity of procedure 1. The computation of $\widetilde{\mathbf{V}}$ (line 1) is $O(mn)$. The inner **for** loop (line 4) performs $O(n)$ operations on lines 6 and 7, and iterates at most m times, once for every neighbor of $\mathbf{i}^+$, adding up to $O(mn)$. Line 3 is also $O(mn)$. The **for** loop on line 2 iterates k (k^* or k_p) times, and is thus $O(kmn)$. Therefore, the total worst-case cost of *COMPUTE_MAX_VDROP* is $O(kmn)$.

In algorithm 1, we pay upfront a one-time-cost $O(mn^2)$ to compute the $\mathbf{b_h}$s by m forward/backward substitutions (line 1). Forming the raise (line 3) can be done in $O(km)$ and the projection in $O(km^2)$, as we will see in sections 5 and 6. Since the number of current sources is far smaller than the number of actual power grid nodes, $m \ll n$, so the most expensive operation in the loop (lines 2-7) is *COMPUTE_MAX_VDROP*, and the total cost, per global constraint, is $O(kmn)$ in the worst-case. For c global constraints, this adds up to $O(ckmn)$. Therefore, the total complexity of algorithm 1 is $O(mn^2 + ckmn)$. Contrast that with the cost of n LPs on $\mathbb{R}^n$. The minimum-raise-computation algorithm of section 5 requires that the local constraints be sorted, which is a standard $O(m \log m)$ operation, that needs to be done only once in algorithm 1, and does not affect the overall complexity.

Finally, algorithm 1 requires storage of the m n-vectors $\mathbf{b_j}$, which can create a memory bottleneck for large n. However, grid locality [5] enables the sparsification of each $\mathbf{b_j}$ by observing that $\mathbf{b_j}$ represents the power grid response to a single current source excitation. Grid locality ensures that the bulk of the response is confined to a relatively small neighborhood surrounding the location of the excitation. Therefore, only a small fraction of the n entries of each $\mathbf{b_h}$ effectively need to be stored.

5. MINIMUM RAISE COMPUTATION

For simplicity of presentation, we assume without loss of generality, that $\dim(\mathcal{G}) = m$, and that $\mathcal{K}$ and $\mathbf{i}$ are a hypercube and a vertex in $\mathbb{R}^m$. If $\dim(\mathcal{G}) = m' < m$, we would be implicitly working in $\mathbb{R}^{m'}$ (see the discussion at the end of section 3.3).

Recall the rank $r(\mathbf{i})$ of a vertex $\mathbf{i}$ of $\mathcal{K}$, defined as the sum of $\mathbf{i}$'s coordinates, and that the j^{th} coordinate of $\mathbf{i}$, $\mathbf{i}(j)$, is either 0 or l_j (section 3.1). Observe that if $\mathcal{G}$ leaves out $\mathbf{i}$, then it must leave out all vertices of rank greater than that of $r(\mathbf{i})$. Clearly, the vertex $\mathbf{i_0} = [l_0\ l_1 \dots l_{m-1}]^T$, with all its coordinates set to the maximum values on the hypercube (i.e., all local constraints ON), is the rank-wise largest vertex of $\mathcal{K}$. Let us denote the j^{th} largest vertex, after $\mathbf{i_0}$, by $\mathbf{i_j}$. We say that j is the *order* of $\mathbf{i_j}$.

The problem becomes to output the sequence $\mathcal{S} = \{\mathbf{i_1}, \dots, \mathbf{i_k}\}$, and set g^* to $r(\mathbf{i_k})$, which would be the minimum value for which k vertices $(\mathbf{i_0}, \dots, \mathbf{i_{k-1}})$ are left out of the raised global constraint. The set $\mathcal{S}$ need not be unique, but we must guarantee that no vertex outside of $\mathcal{S}$ is of strictly larger rank than any vertex in $\mathcal{S}$. If $r(\mathbf{i_k}) = r(\mathbf{i_{k-1}})$, then in order to ensure that k is an upper bound on the number of vertices left out of $\mathcal{G}$, we must find the largest $k' < k$ for which $\mathbf{i_{k'}} > \mathbf{i_{k'-1}}$ (strict inequality), set $g^* = r(\mathbf{i_{k'}})$, and return the largest k' vertices $\{\mathbf{i_0}, \dots, \mathbf{i_{k'-1}}\}$.

Since $\mathbf{i}(j)$ is either 0 or l_j, we can uniquely identify $\mathbf{i}$ by the set of local constraints that are ON in $\mathbf{i}$. The problem becomes to find which constraints to turn ON to obtain k combinations with maximal total sum. This likens our problem to knapsack problems (KP) [6]: we can think of a vertex $\mathbf{i}$ for which l_j is ON as a packing of the knapsack that includes item l_j, and we want to find the k packings with maximal total value. But our problem is different from a usual KP in that we seek k best packings of all the items: a straightforward KP application would solve it with k successive KPs. However, the similarity with KP motivates the use of a dynamic programming, a useful technique for a wide range of KPs [6], in a way that is suited to the present problem.

5.1 High-Level Description And Correctness

We use a table $\mathbf{T}$ each entry of which represents a *unique* vertex of $\mathcal{K}$. $\mathbf{T}$ is initialized with m entries, and the algorithm iterates k times, adding at most one entry to $\mathbf{T}$ in each iteration. The algorithm terminates with less than $(k+m)$ entries in $\mathbf{T}$.

Conceptually, we subdivide the entries of $\mathbf{T}$ into two groups: *non-empty entries* and *empty entries*. Since the total number of vertices of $\mathcal{K}$ is 2^m, the sum of empty and non-empty entries of $\mathbf{T}$ is 2^m – this is for clarity of presentation only: we will not store or work with all 2^m entries at any time, but only with the non-empty entries, which will number at most $k+m$. The table is formed so that the $\mathbf{i_1}, \dots, \mathbf{i_k}$ are represented by non-empty entries. Specifically, there will be a 1-to-1 correspondence between a vertex of $\mathcal{K}$ and a non-empty entry of $\mathbf{T}$: a non-empty entry is *defined as an entry which is "bound to" a vertex* of $\mathcal{K}$ (for convenience, we will also say that the vertex is bound to the non-empty entry). An empty entry could be potentially bound to a number of vertices (based on its column in $\mathbf{T}$, as we will see below), but the exact binding occurs as the empty entry becomes non-empty. In this terminology, binding an empty entry to a vertex means that this entry becomes non-empty. Our algorithm can be viewed as a way of finding a suitable binding for a suitably chosen subset of entries of $\mathbf{T}$. In the description below, we rely extensively on the 1-to-1 correspondence between a non-empty entry and the vertex of $\mathcal{K}$ to which it is bound, which will be formally defined in section 5.4, and we refer to an entry of $\mathbf{T}$ and the vertex it is bound to interchangeably.

We say that a non-empty entry of $\mathbf{T}$, or the vertex to which it is bound, is *ordered*, if its order is known (recall that $\mathbf{i_j}$ has order j). When a vertex is first bound to an entry, it is *unordered*. At any given point in the algorithm, the set of non-empty but unordered entries in $\mathbf{T}$ forms the *frontier* of $\mathbf{T}$, denoted $\mathcal{F}$. We denote by $\mathcal{F}^+$ the non-empty entries which are not on $\mathcal{F}$, i.e., the set of ordered vertices, and by $\mathcal{F}^-$ the remaining vertices of $\mathcal{K}$, which correspond to empty entries in $\mathbf{T}$. Observe that $\mathcal{F} \cup \mathcal{F}^+$ is the set of vertices bound to non-empty entries of $\mathbf{T}$, in short, the set of non-empty entries.

Our aim is to have k entries in $\mathcal{F}^+$. The algorithm works by performing iteratively two steps: 1) *ordering a vertex*, i.e., finding the order of an element of $\mathcal{F}$ bound to a certain vertex of $\mathcal{K}$, then 2) *shifting the frontier*, i.e., binding an entry of $\mathcal{F}^-$ to an unordered vertex, thereby moving it from $\mathcal{F}^-$ to $\mathcal{F}$. Let $\arg\max r(\mathbf{i})$, $\mathbf{i} \in \mathcal{F}$, be a vertex in $\mathcal{F}$ of maximal rank.

Claim 5.1.1 If $\forall \mathbf{i}^- \in \mathcal{F}^-$, $\exists \mathbf{i} \in \mathcal{F}$ such that $r(\mathbf{i}^-) \leq r(\mathbf{i})$, then algorithm 2 returns k rank-wise maximal vertices of $\mathcal{K}$:

Algorithm 2 minimum raise computation (top-level)

```
1: Initialize F and F^-, F^+ = ∅  //Section 5.3
2: while |F^+| = t < k  //Less than k vertices ordered
3:     i_{t+1} = arg max r(i), i ∈ F  //Section 5.3
4:     F^+ = F^+ ∪ {i_{t+1}}
5:     F = F \ {i_{t+1}}  //Delete i_{t+1} from frontier
6:     Shift F  //Section5.4
7: return {i_1, ..., i_k}, g* = r(i_k)
```

PROOF. Follows from the condition that no vertex of $\mathcal{F}^-$ can have a strictly larger rank than a vertex of $\mathcal{F}$ of maximal rank. □

5.2 Table Properties

Let $T_{i,j}$ $(i,j \geq 0)$ denote the entry of $\mathbf{T}$ at row i and column j, and $T_{:,j}$ denote $\mathbf{T}$'s j^{th} column. We impose the following key condition: $T_{i,j}$ can be bound to a vertex $\mathbf{i}$ of $\mathcal{K}$ if and only if $\mathbf{i}(j) = 0$ and $\mathbf{i}(j') = l_{j'}$, $\forall j' > j$. Therefore, the number of columns in $\mathbf{T}$ is m, and entries in $T_{:,j}$ only differ in one or more of their 0^{th} through $(j-1)^{\text{st}}$ coordinates. Note that a vertex may be bound to one column only, corresponding to the position of its rightmost 0 coordinate. Any vertex $\mathbf{i}$ such that $\mathbf{i}(j) = 0$ and $\mathbf{i}(j') = l_{j'}, \forall j' > j$, is said to *belong to* $T_{:,j}$. We denote this by $\mathbf{i} \in T_{:,j}$. Clearly, 2^j vertices belong to $T_{:,j}$. If all entries of $T_{:,j}$ are bound to vertices, thus non-empty, we say that $T_{:,j}$ is *full*. When $T_{:,j}$ is not full, a number of vertices which belong to $T_{:,j}$ are not bound and correspond to empty entries in $T_{:,j}$, or equivalently, to elements in $\mathcal{F}^- \cap T_{:,j}$. The sum of non-empty and empty entries in $T_{:,j}$ is therefore 2^j. Observe that, $\forall T_{i,j}$ (whether empty or not), $i \leq 2^j - 1$. Define the operator $\mathbf{T}\text{–}to\text{–}\mathcal{V}(.)$ which takes a non-empty entry of $\mathbf{T}$ and returns the vertex of $\mathcal{K}$ to which it is bound. For simplicity of notation, we denote $r(T_{i,j}) = r(\mathbf{T}\text{–}to\text{–}\mathcal{V}(T_{i,j}))$, i.e., the rank of the vertex bound to a non-empty entry $T_{i,j}$.

In every column $T_{:,j}$, we preserve the following properties: $\mathcal{T}_1$): If $\mathcal{F} \cap T_{:,j} = \emptyset$, then $T_{:,j}$ is full and $\mathcal{F}^- \cap T_{:,j} = \emptyset$, and $\mathcal{T}_2$): If $T_{i,j} \in \mathcal{F}$, then $\forall \mathbf{i} \in \mathcal{F}^- \cap T_{:,j}$, $r(\mathbf{i}) \leq r(T_{i,j})$, where a vertex $\mathbf{i}$ of $\mathcal{K}$ is in $\mathcal{F}^- \cap T_{:,j}$ if $\mathbf{i} \in T_{:,j}$ but $\mathbf{i}$ is not bound to an entry of $T_{:,j}$.

Claim 5.2.1 $\mathcal{T}_1$ and $\mathcal{T}_2$ guarantee correctness of algorithm 2.

PROOF. $\forall \mathbf{i} \in \mathcal{F}^- \cap T_{:,j}$, by $\mathcal{T}_1$, $\exists i$ such that $T_{i,j} \in \mathcal{F}$. By $\mathcal{T}_2$, $r(\mathbf{i}) \leq r(T_{i,j})$. Correctness follows from claim 5.1.1. □

When an empty entry becomes non-empty, we say this entry is *filled*. For two entries $T_{i_1,j}$ and $T_{i_2,j}$, we say that $T_{i_1,j}$ is NORTH of $T_{i_2,j}$ if $i_1 \leq i_2$, and that $T_{i_2,j}$ is SOUTH of $T_{i_1,j}$ if $i_2 > i_1$.

To satisfy $\mathcal{T}_1$ and $\mathcal{T}_2$, our strategy is to fill every column from NORTH to SOUTH, filling at most a single entry in a single column in every iteration (up to k), such that the non-empty entries are guaranteed to have non-increasing ranks going SOUTH on any column at all times. Let $\mathcal{N}_{i,j} = \{T_{0,j}, \ldots, T_{i,j}\}$ be the set of entries NORTH of $T_{i,j}$, and let $T_{:,j} \setminus \mathcal{N}(i,j)$ be the set of vertices belonging to $T_{:,j}$ save those bound to elements of $\mathcal{N}(i,j)$. Formally, we fill $\mathbf{T}$ upholding this property:

$\mathcal{T}_3$. $\forall T_{i,j}$, if $T_{i+1,j} \in \mathcal{F} \cup \mathcal{F}^+$, then a) $T_{i,j} \in \mathcal{F}^+$, b) $T_{i+1,j}$ is bound to $\arg\max r(\mathbf{i})$, $\mathbf{i} \in T_{:,j} \setminus \mathcal{N}(i,j)$.

Claim 5.2.2 Initializing each $T_{:,j}$ with the rank-wise largest vertex that belongs to it and always upholding $\mathcal{T}_3$ guarantees $\mathcal{T}_1$ and $\mathcal{T}_2$.

Proof. From $\mathcal{T}_3$(a), there may be only one non-empty, unordered entry in any column $T_{:,j}$ which must be its SOUTHMOST non-empty entry. $\mathcal{T}_3$(b) implies that consecutive entries going SOUTH on a column $T_{:,j}$ are bound to a rank-wise maximal unbound vertex which belongs to $T_{:,j}$. Such a vertex must exist, unless $T_{:,j}$ is full ($i = 2^j - 1$) and all its entries ordered. This establishes $\mathcal{T}_1$.

$\mathcal{T}_3$(a) implies that no non-empty entry of $T_{:,j}$ may be SOUTH of any empty entry of $T_{:,j}$. In particular, the entry in $\mathcal{F} \cap T_{:,j}$ must be NORTH of all the empty entries of $T_{:,j}$. $\mathcal{T}_3$(b) gives that no empty entry of $T_{:,j}$ may be bound to a vertex with a strictly larger rank than any non-empty entry of the same column, in particular, the one in $\mathcal{F} \cap T_{:,j}$. This establishes $\mathcal{T}_2$. □

We can shift the frontier $\mathcal{F}$ based on $\mathcal{T}_3$(a): when the SOUTHMOST non-empty entry of a column is ordered, fill the entry below it by binding it to a vertex, which must be unordered and therefore on $\mathcal{F}$. Thus, *the frontier expands SOUTH on every column*. How to choose the vertex with which to form the binding is according to $\mathcal{T}_3$(b), and will be discussed in section 5.4.

For clarity of presentation, we state three direct corollaries to the above. $\mathbf{C}_1$) All non-empty entries of a column are bound to vertices of non-increasing ranks going SOUTH, $\mathbf{C}_2$) Unless a column is full and all its entries ordered, it has one entry and one entry only on the frontier, which is its SOUTHMOST non-empty entry, and $\mathbf{C}_3$) All entries strictly NORTH of the frontier on a column are non-empty and ordered.

5.3 Initialization And Vertex Ordering

Initializing any $T_{:,j}$ with the rank-wise maximum vertex that belongs to it is direct: this is the vertex $\mathbf{i}^+_\mathbf{j}$ such that $\mathbf{i}^+_\mathbf{j}(j) = 0$, $\mathbf{i}^+_\mathbf{j}(j') = l_{j'}, \forall j' \neq j$. Let $T_{i,j} \leftarrow \mathbf{i}$ denote that entry $T_{i,j}$ binds with vertex $\mathbf{i}$. This expands line 1 of algorithm 2:

1: $\mathcal{F}^+ = \mathcal{F} = \emptyset$, $\mathcal{F}^- = \mathbf{T}$
2: **for** $j = 0, \ldots, m-1$
3: $\quad T_{0,j} \leftarrow \mathbf{i}^+_\mathbf{j}$, $\mathcal{F} = \mathcal{F} \cup T_{0,j}$, $\mathcal{F}^- = \mathcal{F}^- \setminus T_{0,j}$

Ordering the next largest vertex into $\mathcal{F}^+$ is equally direct. Let us say that $T_{:,j}$ is *complete* if it is full and all its entries ordered. Expanding lines 2 through 5 of algorithm 2:

1: **while** $|\mathcal{F}^+| = t < k$
2: $\quad$ **for all** $j = 0, \ldots, (m-1)$ such that $T_{:,j}$ is not complete
3: $\quad\quad$ Compare the SOUTHMOST entry of each $T_{:,j}$
4: $\quad\quad$ Select T_{i^*,j^*} with maximal rank //Ties may occur
5: $\quad \mathbf{i_{t+1}} = \mathbf{T}\text{-}to\text{-}\mathcal{V}(T_{i^*,j^*})$, $\mathcal{F}^+ = \mathcal{F}^+ \cup T_{i^*,j^*}$, $\mathcal{F} = \mathcal{F} \setminus T_{i^*,j^*}$

Ties which may occur on line 4 of the above need to be broken by selecting the leftmost entry in the tie (see section 5.4). Since there is at most one entry in $\mathcal{F}$ per column, this choice is unique.

Claim 5.3.1 Let $T_{i_1,j_1} \in \mathcal{F}^+$, and consider some T_{i_2,j_2} a) if $r(T_{i_2,j_2}) > r(T_{i_1,j_1})$, then $T_{i_2,j_2} \in \mathcal{F}^+$, b) if $r(T_{i_2,j_2}) = r(T_{i_1,j_1})$, and if $j_2 < j_1$, then $T_{i_2,j_2} \in \mathcal{F}^+$.

Proof. (a) is immediate, (b) follows from ordering the leftmost entry on $\mathcal{F}$ with maximal rank in a tie. □

5.4 Frontier Shift And Vertex-Entry Binding

We assume, without loss of generality, that $l_0 \leq \ldots \leq l_{m-1}$ (see section 4.1). Recall that we can denote a vertex $\mathbf{i}$ as the union set of all its non-zero coordinates. and that only the coordinates corresponding to $l_0, \ldots, l_{j-1}$. may change among different vertices belonging to $T_{:,j}$. The following is easy to see:

Claim 5.4.1 $\forall \mathbf{i_1} \in T_{:,j_1}$, $j_1 > 0$, $\exists!$ $\mathbf{i_2} \in T_{:,j_2}$, $j_2 < j_1$, such that $\mathbf{i_1} = \mathbf{i_2} \setminus \{l_{j_1}\}$. □

In the above, we say that $\mathbf{i_1}$ *is paired with* $\mathbf{i_2}$, and write $\mathbf{i_2} = \Pi(\mathbf{i_1})$. Note that $\mathbf{i_1}$ and $\mathbf{i_2}$ share all their coordinates except the j_1^{st}: $\mathbf{i_2}(j_1) = l_{j_1}$, $\mathbf{i_1}(j_1) = 0$. In this perspective, observe that the initial entry $\mathbf{i}^+_\mathbf{j}$ of any $T_{:,j}$ (section 5.3) is paired with $\mathbf{i_0}$. Note that a vertex is fully determined by the column to which it belongs and the vertex with which it is paired.

A similar description can be given in terms of the entries of $\mathbf{T}$. If T_{i_1,j_1} and T_{i_2,j_2} are bound to $\mathbf{i_1}$ and $\mathbf{i_2}$, the above claim can be viewed as forming T_{i_1,j_1} by pairing $T_{:,j_1}$ with T_{i_2,j_2}, $j_2 < j_1$. For convenience, we simply say that T_{i_1,j_1} *is paired with* T_{i_2,j_2} and write $T_{i_2,j_2} = \Pi(T_{i_1,j_1})$. We can thus define $\mathbf{T}\text{-}to\text{-}\mathcal{V}(T_{i,j})$ recursively, as follows:

$$\mathbf{T}\text{-}to\text{-}\mathcal{V}(T_{i,j}) = \begin{cases} \mathbf{T}\text{-}to\text{-}\mathcal{V}(T_{i,j}) = \mathbf{i}^+_\mathbf{j} & \text{, if } i = 0 \\ \mathbf{T}\text{-}to\text{-}\mathcal{V}(\Pi(T_{i,j})) \setminus \{l_j\} & \text{, otherwise} \end{cases} \quad (14)$$

Notice that pairing some entry $T_{i+1,j}$ with another entry $T_{i',j'}$, bound to a vertex $\mathbf{i'}$, is equivalent to binding $T_{i+1,j}$ to $\mathbf{i'} \setminus \{l_j\}$. As $T_{i+1,j}$ is paired, it gets filled and becomes part of $\mathcal{F}$: *the frontier shifted into* $T_{i+1,j}$. Therefore, selecting the vertex to which $T_{i+1,j}$ is bound is equivalent to selecting the entry $T_{i',j'}$ with which to pair $T_{i+1,j}$. The question becomes how to select $T_{i',j'}$ in a way that satisfies $\mathcal{T}_3$(b).

Let $r_0 = \sum_{j=0}^{m-1} l_j$, we can write:

$$r(T_{i,j}) = r_0 - l_j - (r_0 - r(\Pi(T_{i,j}))) = r(\Pi(T_{i,j})) - l_j, \quad (15)$$

which immediately leads to:

Claim 5.4.2 $r(T_{i_1,j}) \leq r(T_{i_2,j})$ if and only if $r(\Pi(T_{i_1,j})) \leq r(\Pi(T_{i_2,j}))$. □

We can now compute the appropriate vertex to bind to a table entry $T_{i+1,j}$, satisfying $\mathcal{T}_3$(b). We assume that 1) all entries of $\mathcal{N}(i,j) = \{T_{0,j}, \ldots, T_{i,j}\}$ are non-empty and ordered, and 2) we are looking to shift the frontier into $T_{i+1,j}$.

Let $\Pi(\mathcal{N}(i,j)) = \bigcup \Pi(T_{i',j'})$, $T_{i',j'} \in \mathcal{N}(i,j)$, be the set of table entries with which the elements of $\mathcal{N}(i,j)$ are paired. Let $\arg\max r(T_{i,j})$ be a table entry bound to a vertex of maximal rank. The following is key:

Corollary 2. $\Pi\left(\arg\max_{\mathbf{i} \in T_{:,j} \setminus \mathcal{N}(i,j)} r(\mathbf{i})\right) = \mathbf{T}\text{-}to\text{-}\mathcal{V}(T_{i_p,j_p})$, *where* $T_{i_p,j_p} \in \mathcal{F}^+$ *and is given by:*

$$T_{i_p,j_p} = \arg\max r(T_{i',j'}),\ j' < j,\ T_{i',j'} \notin \Pi(\mathcal{N}(i,j)). \quad (16)$$

Proof. (16) follows from claim 5.4.2. Note that, if $T_{i_p,j_p} \in \Pi(\mathcal{N}(i,j))$, then pairing $T_{i+1,j}$ with it means binding a vertex to $T_{i+1,j}$ that is already bound to some entry in $\mathcal{N}(i,j)$. We still must prove that $T_{i_p,j_p} \in \mathcal{F}^+$.

Let $T^-_j = \{T_{:,0}, \ldots, T_{:,j-1}\}$. We prove that $T_{i_p,j_p} \in \mathcal{F}^+$ by showing that a) $\exists T_{i_1,j_1} \in (\mathcal{F}^+ \cap T^-_j) \setminus \Pi(\mathcal{N}(i,j))$, and b) T_{i_p,j_p} never has to be necessarily in $\mathcal{F}$.

Let $\Pi(T_{i,j}) = T_{i_0,j_0}$. From (15), $r(T_{i,j}) < r(T_{i_0,j_0})$. Since $T_{i,j} \in \mathcal{F}^+$, then by claim 5.3.1, $T_{i_0,j_0} \in \mathcal{F}^+$. We distinguish two cases: 1) $j_0 < j-1$, and 2) $j_0 = j-1$.

Case 1: Consider $T_{i_1,j-1}$ such that $\Pi(T_{i_1,j-1}) = T_{i_0,j_0}$. Because $l_{j-1} \leq l_j$, $r(T_{i_1,j-1}) \geq r(T_{i,j})$, so $T_{i_1,j-1} \in \mathcal{F}^+$ (see claim 5.3.1). Also, $r(T_{i_1,j-1}) < r(T_{i_0,j_0})$. Since $\Pi(T_{i,j}) = T_{i_0,j_0}$, then if $T_{i_1,j-1} \in \Pi(\mathcal{N}(i,j))$, we would have a contradiction with (16).

Case 2: Let $\mathbf{i}$ be the vertex bound to T_{i_0,j_0} ($j_0 = j-1$). Unless $T_{:,j}$ is complete (cannot fill $T_{i+1,j}$), $\exists j' < j_0$ such that $\mathbf{i}(j') = l_{j'}$. Letting $\mathbf{i}(j') = 0$ yields a vertex $\mathbf{i'} \in T_{:,j_0}$ such

that $r(\mathbf{i}') \geq r(T_{i,j})$. By claim 5.3.1, $\mathbf{i}'$ must be bound to some $T_{i_1,j_0} \in \mathcal{F}^+$. Also, $r(T_{i_1,j_0}) < r(T_{i_0,j_0})$, implying that $T_{i_1,j_0} \notin \Pi(\mathcal{N}(i,j))$, to not contradict (16).

Cases 1 and 2 establish a). From a) and claim 5.3.1, $r(T_{i_1,j_1}) \geq r(T_{i_2,j_2}), \forall T_{i_2,j_2} \in \mathcal{F}$. This establishes b). □

Let $\Phi(T_{i,j})$ denote the order of (the vertex bound to) $T_{i,j}$, and $\Phi^{-1}(x)$ the table entry (or the vertex bound to it) of order x. As a result of corollary 2, if T_{i^*,j^*} is ordered at iteration t (section 5.3), and say T_{i^*,j^*} is paired with some T_{i_0,j_0}, and we want to shift the frontier on $T_{:,j^*}$, we pair T_{i^*+1,j^*} with the entry of *smallest order* greater than $\Phi(T_{i_0,j_0})$, which must be of maximal rank less than $r(T_{i_0,j_0})$, and which belongs to a column less than j^*. Thus, we can expand line 6 of algorithm 2 as:

1: **if** $T_{:,j^*}$ is not complete **then**
2: $\quad T_{i_1,j_1} = \mathbf{fsog}(T_{i^*,j^*})$
3: $\quad$ Fill T_{i^*+1,j^*} such that $\Pi(T_{i^*+1,j^*}) = T_{i_1,j_1}$
4: $\quad \mathcal{F} = \mathcal{F} \cup T_{i^*+1,j^*}$, $\mathcal{F}^- = \mathcal{F}^- \setminus T_{i^*+1,j^*}$

fsog returns the table entry T_{i_1,j_1} of *smallest order* greater than $x = \Phi(\Pi(T_{i^*,j^*}))$ which is in columns left of $T_{:,j^*}$. A key observation is that $\Phi(T_{i_1,j_1}) \geq x+1$. Therefore, we may first *attempt to pair* T_{i^*+1,j^*} with $T_{i_1,j_1} = \Phi^{-1}(x+1)$. If $j_1 < j^*$, the pairing is successful. Otherwise, we could traverse $\mathcal{F}^+$ by vertices of increasing order, starting at T_{i_1,j_1} until reaching the first vertex left of $T_{:,j^*}$. But this can be avoided with some bookkeeping. Let $T_{i^{(t)},j^{(t)}}$ be the entry bound to $\mathbf{i_t}$ ($T_{i^*,j^*} := T_{i^{(t+1)},j^{(t+1)}}$), and define $\mathbf{fsog}(T_{i^*,j^*})$ as follows:

1: **if** $j^* \geq j^{(t)}$ **then**
2: $\quad$ **if** *needs_next_pairing*$(j^{(t)})$ == **false then**
3: $\quad\quad$ *needs_next_pairing*$(j^{(t)})$ = **true**
4: $\quad\quad$ *needs_next_pairing_at_row*$(j^{(t)}) = i^{(*,t)}$
5: **for all** $j = j^*+1, \ldots, m-1$
6: $\quad$ **if** *needs_next_pairing*(j) == **true then**
7: $\quad\quad$ i = *needs_next_pairing_at_row*(j)
8: $\quad\quad$ *jump_to*$(T_{i,j}) = t+1$
9: $\quad\quad$ *needs_next_pairing*(j) = **false**
10: *jump_to*$(T_{i^*,j^*}) = t+2$
11: $T_{i_1,j_1} = \Phi^{-1}\left(\Phi(\Pi(T_{i^*,j^*}))+1\right)$
12: **if** $j_1 < j^*$: **return** T_{i_1,j_1}
13: **else**: t' =*jump_to*(T_{i_1,j_1}); **return** $T_{i^{(t')},j^{(t')}}$

Clearly, traversal of $\mathcal{F}^+$ would be $O(t)$ in iteration t, for a total of $O(k^2)$ for the k iterations. The above procedure does the job in $O(m)$ per iteration (line 5, everything else is $O(1)$), for a total of $O(mk)$, by storing in *jump_to*$(T_{i,j})$ the order of the entry to which to jump directly, should a pairing between an entry of $T_{:,j}$ and $T_{i,j}$ be attempted. We omit a detailed proof of its correctness noting that it follows from the fact that, by corollary 2, an ordered vertex of maximal rank exists in columns less than j^* when a pairing between two entries of $T_{:,j^*}$ is attempted.

5.5 Complexity And Sample Run

Clearly, the initialization phase is $O(m)$ (section 5.3). Vertex ordering is $O(m)$ *per iteration* (section 5.3). All operations in frontier shifting are $O(1)$, except **fsog** which is $O(m)$ *per iteration* (section 5.4). Therefore, the total cost is $O(mk)$, to fill the table with k entries bound to vertices of maximal ranks. Next we need to compute $\mathbf{i_k}$ (line 7) of algorithm 2. By outputting $\mathbf{i_1}, \ldots, \mathbf{i_k}$ sequentially, a vertex is always output *after* the vertex with which it is paired, so that $\mathbf{T}-to-\mathcal{V}$ becomes $O(m)$ *per vertex*, for a total $O(mk)$, which is the asymptotic complexity of the algorithm. In terms of memory footprint, it is clear that we only need to store non-empty entries: at the end of the algorithm, we have k ordered entries and at most m on the frontier, i.e., $O(k+m)$. For reference, a sample run of algorithm 2 is shown in table 1.

6. CONSTRAINT PROJECTION

If $\mathcal{I}$ is the set of indices of current sources included in a global constraint $\mathcal{G} : \sum_{j\in\mathcal{I}} i_j \leq g$, we are interested in finding a set $\mathcal{I}_p \subset \mathcal{I}$ such that the projected constraint $\mathcal{G}_p : \sum_{j\in\mathcal{I}_p} i_j \leq g$ leaves out at most k vertices of $\mathcal{K}$, and compute these vertices.

Table 1: Sample run of algorithm 2. For every entry is shown a pair (Π, Φ), where Π is the entry with which it is paired and Φ its order, and the vertex bound to it, as the union of its non-zero coordinates.

$l_0 = 1$	$l_1 = 2$	$l_2 = 2$	$l_3 = 2.5$
$(\mathbf{i_0}, 1)$ $\{l_1, l_2, l_3\}$	$(\mathbf{i_0}, 2)$ $\{l_0, l_2, l_3\}$	$(\mathbf{i_0}, 3)$ $\{l_0, l_1, l_3\}$	$(\mathbf{i_0}, 6)$ $\{l_0, l_1, l_2\}$
Complete	$(T_{0,0}, 4)$ $\{l_2, l_3\}$	$(T_{0,0}, 5)$ $\{l_1, l_3\}$	$(T_{0,0}, 8)$ $\{l_1, l_2\}$
	Complete	$(T_{0,1}, 7)$ $\{l_0, l_3\}$	$(T_{0,1}, \mathcal{F})$ $\{l_0, l_2\}$
		$(T_{1,1}, \mathcal{F})$ $\{l_3\}$	

Table 2: Results on small grids where comparison with the exact solution is practical. The exact solution was computed based on 30 time steps.

Grid size (#nodes)	# Current sources	Runtime (exact solution)	Runtime (proposed method)	Average overestimation (%)
94	20	3 min.	$<$ 1 sec.	5.64
124	20	5 min.	$<$ 1 sec.	6.31
391	90	1.1 hrs.	1.5 sec.	6.02
612	110	4 hrs.	4 sec.	4.12

We say that $\mathcal{G}_p$ is *formed over* $\mathcal{I}_p$. Note that g is the *value* of both $\mathcal{G}$ and $\mathcal{G}_p$. Our strategy is to form several global constraints of value g over different subsets of $\mathcal{I}$ and to select among them one or many to form the projection.

Let $\mathcal{I}_{p_1}$ and $\mathcal{I}_{p_2}$ be two distinct subsets of $\mathcal{I}$, and let $\mathcal{G}_{p_1}$ and $\mathcal{G}_{p_2}$ be the global constraints formed over them, respectively. In general, the local constraints corresponding to currents in $\mathcal{I}_{p_1}$ and $\mathcal{I}_{p_2}$ differ and so must the local constraint hypercubes corresponding to $\mathcal{G}_{p_1}$ and $\mathcal{G}_{p_2}$. We make this explicit by denoting by $\mathcal{K}(\mathcal{G}_p)$ the hypercube formed by the local constraints included in $\mathcal{I}_p$. Similar to section 5, we let $\mathbf{i_k}(\mathcal{G}_p)$ denote a k^{th} rank-wise maximal vertex of $\mathcal{K}(\mathcal{G}_p)$ left out by $\mathcal{G}_p$.

Algorithm 2 is our workhorse. Recall the rank $r(\mathbf{i}) = \sum_{j\in\mathcal{I}_p} \mathbf{i}(j)$ of a vertex $\mathbf{i}$ ($j \in \mathcal{I}_p$ was implicit in previous sections). Observe that $\mathcal{G}_p$ leaves out at most k vertices of $\mathcal{K}(\mathcal{G}_p)$ if and only if $r(\mathbf{i_k}(\mathcal{G}_p)) < g$. Therefore, for any $\mathcal{I}_p$, we can run algorithm 2 and output the k rank-wise largest vertices of $\mathcal{K}(\mathcal{G}_p)$, and compute $r(\mathbf{i_k}(\mathcal{G}_p))$.

Our approach is to start with $\mathcal{I}_p = \emptyset$ and add one current source to $\mathcal{I}_p$ at a time until $r(\mathbf{i_k}(\mathcal{G}_p)) \geq g$, invoking algorithm 2 every time. The next current source to add to $\mathcal{I}_p$ can be chosen in many ways. We propose two such possibilities: *highest-to-lowest*: add the current source with the largest local constraint not included so far. This would include in $\mathcal{G}_p$ many potential big offenders (e.g., I/O pads, clock pins) among the currents in $\mathcal{I}$, and *lowest-to-highest*, which works the other way, and would result in $\mathcal{I}_p$ including a large numbers of currents originally in $\mathcal{I}$. In terms of complexity, every run of algorithm 2 is $O(k|\mathcal{I}_p|)$. If $|I_p| = 1$ up to $m-1$ at most, we end up with $O(km^2)$, in line with algorithm 1.

7. EXPERIMENTAL RESULTS

To test our method, we wrote a C++ package that enables us to generate power grids from user specifications, including grid dimensions, metal layers (M1 – M9), pitch and width per layer, and C4 and current source distribution. A global constraint is specified by the spatial region and metal layers it includes. All the reported results are on grids with seven global constraints covering the entirety of the grid area, and runtimes follow the application of all seven global constraints. Minimum spacing and sheet and via resistances were specified according to a 90 nm technology. All results were obtained on a 1.1 GHz Sun machine, with 4 GB of memory.

Table 2 shows the accuracy of our proposed technique for estimating the vector of voltage drop upper bounds, by comparing

Table 4: Prototyping of various grid configurations.

Configuration	Base configuration	Layers	Modifications from base	# Nodes	Max voltage drop	Location	Runtime
B_0	–	M1–M6	–	570,000	4% V_{dd}	(3.6, 2.7) on M1	1.5 hrs.
B_1	B_0	M1–M6	pitch M2 ×2	357,000	12% V_{dd}	(3.6, 2.25) on M1	47 min.
B_2	B_0	M1–M5	pitch M2 ×1.5 pitch M3,M4 ×2.3 width M3 ×0.6 width M4 ×0.45 width M5 ×0.36	263,000	7.5% V_{dd}	(3.6, 2.25) on M1	30 min.
B_3	B_2	M1–M5	pitch M2 ×2	170,000	9.9% V_{dd}	(3.15, 2.25) on M1	27 min.
B_4	B_0	M1–M5	width M1,M2 ×0.9 width M3,M4 ×1.2 width M5 ×1.5	438,000	5% V_{dd}	(3.6, 2.7) on M1	55 min.
B_5	B_4	M1–M5	pitch M2 ×2	245,000	13% V_{dd}	(3.6, 2.25) on M1	30 min.

Table 3: Accuracy and speed comparisons of the proposed approach with sequential linear programs.

Grid size (#nodes)	Analysis type	Runtime (sequential LPs)	Runtime (proposed method)	Average overestimation(%)
2,380	TR	30 min.	7 sec.	4.39
3,780	TR	2 hrs.	11 sec.	4.78
24,000	DC	2.4 min./node (est. 40 days)	5 min.	5.55 (est.)
41,000	DC	6.5 min./node (est. 187 days)	9.5 min	6.02 (est.)

our estimates with the vector of exact voltage drop maxima at the 30^{th} time step, given the current constraints, Computing the exact maximum voltage drop vector at the q^{th} time step is done as in section 5.1 of [1], and involves solving one LP per node, over $\mathbb{R}^{nq}$, where n is the size of the power grid. Clearly, this computation is extremely expensive, as it increases the dimension of each of the n LPs by a factor of q, and can only serve as a benchmarking measure for small grids. Besides, it has the inherent shortcoming of being able to handle a finite, and relatively small, number of time steps, whereas the upper bound applies at $t \to \infty$. Column 5 shows the average accuracy of the estimate, taken over the n nodes. These numbers show a relatively small average overestimation following our approach, which could only diminish if we were to compute the exact voltage drops over more than 30 steps. Clearly, the proposed approach runs several orders of magnitude faster than the exact computation.

Table 3 compares results of the proposed method with the execution of n LPs sequentially, one LP per maximum node voltage, as in [1]. The top three rows are based on transient analysis (TR), the bottom three DC. The number of current sources ranged from 110 to 270, and k^* and k_p were 1000. For transient analysis, column 3 refers to the time it took to compute $\mathbf{V_u}$ (upper bounds on node voltage drops), which is essentially the cost of computing $\mathbf{V_a}$ by n LPs (see the corollary of theorem 1). Since the full vector $\mathbf{V_a}$ is needed for the computation of $\mathbf{V_u}$, we had to solve all n LPs, which limited the size of grids on which this comparison could be made. Accuracy estimates were obtained on larger grids through DC analysis, where we could test our method on a node-by-node basis, without the need for all n LP solutions. For this purpose, we chose 10 random nodes, solved the 10 corresponding LPs, and estimated runtime (column 3) and accuracy (column 5) based on these 10 nodes. Given the impractical runtimes of sequential LPs, our method simply makes checking a power grid under uncertain, constraint-specified currents, a feasible and practical proposition.

Table 4 shows how our method can be used for prototyping various grid configurations. Starting with a (5mm × 5mm) power grid, and a power budget of 1.28 W non-uniformly distributed over 156 current sources, we test out six different configurations by varying the number of metal layers and the widths and pitches of some layers. Each configuration has a base configuration (column 2), and the deviations between the two configurations are summarized in column 4. Using the proposed technique, we are able to visualize maximum voltage drop anywhere on the grid. For illustration, we only show the worst-case maximum drop (column 6) and its location (column 7), in every configuration.

8. CONCLUSION

As power grid safety becomes increasingly important, so does the need to start power grid verification early in the design cycle and incorporate circuit uncertainty of the early stages into useful power grid information. We resort to the framework of capturing circuit uncertainty via constraints on currents and maximizing node voltage drops over the constraint space, and propose a geometric approach to transform a problem whose solution requires as many linear programs as there are nodes, to another involving a user-limited number of solutions of a single linear system. The key is to determine, from the geometry of the problem, a limited number of points in the current space at which to solve the power grid system, and to derive estimates of the power grid voltage drops from the solutions. This approach made the problem of verifying full power grids under uncertain currents practicable and scalable. Prior art simply couldn't handle grids of more than a few hundred or thousand nodes, suffering too prohibitive runtimes for any larger systems. This improves runtime manyfold, doing in seconds what required hours, and finishing the verification of grids of about half a million nodes in 1–1.5 hours. Our approach is designed so that all approximation is conservative, and our results showed inaccuracy was relatively small, rarely exceeding 6% on average.

Acknowledgment

The authors thank Dr. U. Pferschy from the University of Graz, and R. Beidas and H. Mangassarian from the University of Toronto for their helpful comments on the algorithm in section 5.

9. REFERENCES

[1] M. Nizam, F. N. Najm, and A. Devgan. Power grid voltage integrity verification. In *ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 239–244, San Diego, CA, August 8-10 2005.

[2] I. A. Ferzli, F. N. Najm, and L. Kruse. Early power grid verification under circuit current uncertainties. In *ACM/IEEE International Symposium on Low Power Electronics and Design*, Portland, OR, August 27-29 2007.

[3] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[4] R. Horst, P. P. Pardalos, and N. V. Thoai. *Introduction to Global Optimization*. Kluwer, 2000.

[5] E. Chiprout. Fast flip-chip power grid analysis via locality and grid shells. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 485–488, San Jose, CA, November 7-11 2004.

[6] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.