A Graph Based Simplex Method for the Integer Minimum Perturbation Problem with Sum and Difference Constraints

Alexey Lvov and Fook-Luen Heng
IBM T.J. Watson Research Center, Yorktown Heights, NY, 10598, USA
{lvov@us.ibm.com heng@us.ibm.com}

ABSTRACT

The integer minimum perturbation problem with sum and difference constraints is stated as follows: minimize $f(x) = a_1|x_1 - b_1| + a_2|x_2 - b_2| + \ldots + a_n|x_n - b_n|$ under constraints

$$\begin{array}{c} \pm x_{i_1} \pm x_{j_1} \geq c_1, \\ \pm x_{i_2} \pm x_{j_2} \geq c_2, \\ \ldots \quad \ldots \quad \ldots \\ \pm x_{i_m} \pm x_{j_m} \geq c_m, \end{array}$$

where the sign in front of each variable is either "+" or "−", $a_1, a_2, \ldots, a_n \geq 0$ and all variables and constants are integers.

The minimum perturbation problem [6] arose from layout migration. The sum and difference constraints arose from the hierarchical nature of the layout. We proposed and implemented a graph based algorithm to solve this optimization problem. Our algorithm consists of two steps. First find the optimal solution for the non-integer version of the problem by using a modification of simplex method which takes advantage of the special form of the constraints (a graph based simplex method). Then find an integer solution close to the optimal by solving a 2-SAT problem. The time complexity of the algorithm is $O\big(p(m+n)\big)$, where p is the number of pivots in the simplex algorithm; note that the regular simplex method, being applied to this problem, would require $O(pn(m+n))$ time.

Our result on production layouts shows that the runtime scale very well with a $O(nlog(n))$ scanline algorithm used to generate the constraints for the layouts. This makes it a very practical solver for the problem.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: [Linear Programming, Integer Programming]; J.6 [**Computer-Aided Engineering**]: Comp - uter-aided design

General Terms

Algorithms, Design.

Keywords

Linear Programming, Optimization, Design Migration.

1. INTRODUCTION

Layout optimization techniques have been studied in the literature in several contexts. The traditional symbolic layout to physical layout translation takes the form of compaction followed by wire-length minimization [8]. In yield enhancement, some parts of a layout are frozen and wires are spread apart. In design migration, the problem is formulated as a minimum perturbation problem [6]. In some specific scenarios, such as electromigration reliability enhancement, a special algorithm is developed to speed up the layout optimization process [3]. More recently altPSM compliance layout is legalized in the same layout optimization framework [5] [9]. All the layout optimization techniques aforementioned use a constraint graph [8] to capture the design ground rules requirement to ensure the legality of the final layout. Since all layout units need to conform to the basic technology unit, e.g. 0.01u for the 90nm technology node, the layout units are represented as integer units in a layout system. A valid solution to the layout optimization problem needs to be an integer solution.

A constraint graph is a directed graph which represents a set of 2-variable difference constraints. Each directed arc or each 2-variable diff. constraint $x_i - x_j \geq d_{ij}$ represents a distance requirement between two adjacent layout elements.

In the presence of more complex layout constraints, such as hierarchical constraints and symmetry constraints, a layout will have to be modeled by more general linear constraints. The layout optimization problem then becomes a more general Integer Linear Programming (ILP) problem [7] [11].

An $O(mU)$ time, where U is the range of integers, algorithm for 2-variable constraints problem is presented in [2]. This is not suitable for our application since the integer range in our layout problem is typically in the millions.

A special class of ILP problems which have a small number of general linear constraints has been investigated in [13]. In this paper we investigate a graph based solution for another class of layout ILP problems, the problems which have an arbitrary set of 2-variable sum and difference constraints. This class of problems can arise in hierarchical layout (i) and in layout with symmetry constraints (ii).

i) In practice, large layouts such as cores and large macros are described hierarchically for clarity and efficiency. Suppose a cell C_1 is instantiated through a sequence of transformations $T_n \circ T_{n-1} \circ \ldots \circ T_1$, called the *instance path* of C_1. Cell C_1 is transformed in its parent cell C_2 by transformation T_1, C_2 is transformed in its parent cell C_3 by T_2 and so on, T_n is the transformation from the the cell next to the root (the top cell of the path) to the root. $T_1, T_2, \ldots, T_n \in G$, where G is the group generated by the symmetry transformations of the unit square (8 transf.) and all parallel shifts.

Suppose all the transformations are given and are not allowed to change during the optimization. Let design edges A and B be parallel and be bound by some distance constraint. Without loss of generality assume that A, B are vertical and that the constraint has form $X_A - X_B \geq d$. Note that our group of transformations G is generated by all "$\pm t + \beta$" transformations applied to "x" and "y" coordinates independently and a single additional transformation of swapping the "x" and "y" coordinates. The world (or root) x-coordinates X_A, X_B of our edges can be expressed via their coord. in their leaf cells in the following way (depending on whether the numbers of "x-y swaps" in the transformation paths are even or odd there are four combinations):

$\begin{pmatrix} X_A \\ \star \end{pmatrix} = T_n \circ T_{n-1} \circ \ldots \circ T_1 \begin{pmatrix} x_A \\ \star \end{pmatrix}$ or $\begin{pmatrix} X_A \\ \star \end{pmatrix} = T_n \circ T_{n-1} \circ \ldots \circ T_1 \begin{pmatrix} \star \\ y_A \end{pmatrix}$,

$\begin{pmatrix} X_B \\ \star \end{pmatrix} = S_m \circ S_{m-1} \circ \ldots \circ S_1 \begin{pmatrix} x_B \\ \star \end{pmatrix}$ or $\begin{pmatrix} X_B \\ \star \end{pmatrix} = S_m \circ S_{m-1} \circ \ldots \circ S_1 \begin{pmatrix} \star \\ y_B \end{pmatrix}$.

This gives one of the following 4 types of constraints $\pm x_A \pm x_B \geq const$, $\pm x_A \pm y_B \geq const$, $\pm y_A \pm x_B \geq const$, $\pm y_A \pm y_B \geq const$ on the local coord. of A and B in their cells.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

Therefore, a hierarchical layout optimization problem in which the translation factors are fixed a priori results in a 2-variable sum and difference constraint problem.

ii) The 2-variable sum constraints also arise in the presence of symmetry constraints. For example, if two layout objects x_i and x_j are required to be at equal distance from a fixed location L, it can be expressed as follows: $x_i + x_j \geq 2L$, $x_i + x_j \leq 2L$, $x_i \leq L$, $x_j \geq L$.

The algorithm we describe in this paper can be used to solve any linear optimization problem with a convex piecewise-linear objective function with sum-and-difference 2-variable constraints (*break hyperplanes*).

For the purpose of illustration we use the minimum perturbation objective.

Our algorithm consists of two steps. **At the first step** we find an[1] optimal solution x_{opt} to the problem in rational numbers. This step is described in Sections 2, 3.

THEOREM 1.1. *Assume that an optimal rational solution x_{opt} exists. If the convex polyhedron formed by the feasible rational solutions of integer minimum perturbation problem $\mathcal{P}$ has a non-zero volume then $\mathcal{P}$ has a feasible integer solution x_{int} such that L_∞ distance[2] $\rho_\infty(x_{opt}, x_{int}) \leq 1/2$ (call such solution "$\frac{1}{2}$-near-optimal").*

PROOF: We prove this theorem in Section 4. □

So, a non-degenerate space of feasible rational solutions always contains a $\frac{1}{2}$-*near-optimal* solution. Note that the condition of non-degeneracy is important. For example problem $f(x) = |x_3|$, $1 \geq x_1 + x_2 \geq 1$, $0 \geq x_1 - x_2 \geq 0$, $x_3 + x_3 \geq 1$, has optimal rational solution $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$, but does not have any feasible integer solution.

At the second step of the algorithm, given an optimal rational solution x_{opt}, we find a $\frac{1}{2}$-near-optimal solution. The process of finding a $\frac{1}{2}$-*near-optimal* solution reduces to solving a 2-SAT problem (see Section 4). Note that, unlike general SAT, 2-SAT belongs to the class of problems solvable in polynomial time (a nice algorithm for 2-SAT is given, for example, in [12] pp. 184-185 or [1]).

THEOREM 1.2. *If at least one $\frac{1}{2}$-near-optimal solution exists, our algorithm always, independently of whether or not the condition of Theorem 1.1 is satisfied, establishes this fact and produces a $\frac{1}{2}$-near-optimal solution. Otherwise it establishes that there is no $\frac{1}{2}$-near-optimal solution.*

PROOF: We prove this theorem in Section 4. □

Now Theorem 1.1 and Theorem 1.2 completely describe the gist of our algorithm.

Though this algorithm does not guarantee that $f(x_{int})$ is the minimum possible value of the objective function over all feasible integer arguments, it far outperforms any algorithm achievable for the general integer linear programming problem where

- even for a non-degenerate feasible polyhedron an optimal integer solution does not necessarily exists,
- if a solution exists it can be arbitrarily far from any optimal rational solution,
- finding even a $\frac{1}{2}$-*near-optimal* solution x_{int} is NP-hard.

In Section 5 we give analysis of the time complexity of the algorithm illustrated with a detailed example.

2. A GRAPH REPRESENTATION OF "SUM AND DIFFERENCE" LINEAR SYSTEMS

DEFINITION 2.1. *Call linear system $Ax = c$ a Sum-and-Difference system if all elements of A,c are integers and the sum of the absolute values of elements in each row of A is equal to 2. For example*

$$\begin{bmatrix} 1 & 1 & \circ & \circ & \circ & \circ \\ -1 & \circ & \circ & -1 & \circ & \circ \\ 1 & \circ & \circ & \circ & 1 & \circ \\ \circ & -1 & 1 & \circ & \circ & \circ \\ \circ & \circ & -1 & -1 & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 16 \\ 20 \\ 24 \end{bmatrix}.$$

With each Sum-and-Difference system we associate a graph. The vertices of the graph correspond to the unknowns of the system, the edges correspond to the equations. The graph has two types of edges: red - for equations with one "1" and one "-1" and black - for equations with "...1...1..." or "...-1...-1...". A black edge can connect a vertex to itself which corresponds to a single "2" or "-2". The graph is completely determined by the matrix of the system, it contains some, but not all, information about the system (see Figure 1).

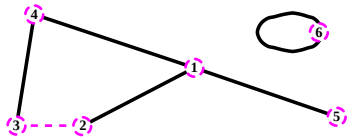


Figure 1: The red-black graph of the sys. in the ex. "Red" edges are shown by gray dashed lines.

LEMMA 2.2. *The rows of Sum-and-Difference matrix A form a basis if and only if each connected component of the corresponding red-black graph has exactly one elementary cycle[3] and each such cycle has an odd number of black edges.*

PROOF: We omit the proof for the purpose of this extended abstract. □

3. FINDING AN OPTIMAL SOLUTION IN RATIONAL NUMBERS

In this section we present an algorithm for solving the minimum perturbation problem in rational numbers. Our algorithm uses a classical linear programming technique called Simplex Method, described in detail, for example, in [10]. We omit all proofs related to the correctness of Simplex Method itself and only describe the modification of this method that applies to the minimum perturbation problem.

Extend the notion of the red-black graph introduced in the previous section so that the graph contains all the information about the problem:

For a "$x_i - x_j \geq c_0$"-type constraint draw a directed red edge from x_j to x_i and assign it a (symbolic) weight "$M($"c_0" $+ b - e)$" (see Figure 2).

Figure 2: "b" stands for the *beginning* of the edge, "e" stands for the *end* of the edge.

For a "$x_i + x_j \geq c_0$"-type constraint draw a black edge between x_i and x_j and assign it a (symbolic) weight "$M($"c_0"$- x - x)$" (see Figure 3).

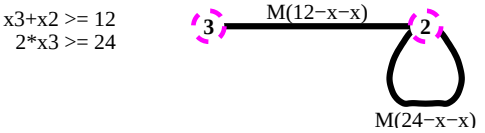


Figure 3: The two "x"-es stand for the two ends of the edge.

[1]Sometimes it can be more then one optimal solution, for example optimal solutions may form a segment with the same value of $f(x)$ at each point.

[2]L_∞ norm is the maximum of the absolute values of the coordinates.

[3]An elementary cycle is a cycle whose vertices $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ are all different (see [4], pp. 4-7). Note that, in particular, two different edges connecting the same pair of vertices form an elementary cycle.

For a "$-x_i - x_j \geq c_0$"-type constraint draw a black edge between x_i and x_j and assign it a (symbolic) weight "M("c_0"$+x+x$)". For an "$a_i|x_i - b_i|$" item in the formula for $f(x)$ draw two black x_i-loops and assign them (symbolic) weights a_i"($2b_i$"$-x-x$)" and a_i"($-2b_i$"$+x+x$)".

We minimize continuous piecewise-linear convex function $f(x)$ subject to the set of linear constraints

$$\begin{aligned} l_1(x) &\stackrel{\text{def}}{=} c_1 - (\pm x_{i_1} \pm x_{j_1}), \quad l_1(x) \leq 0, \\ l_2(x) &\stackrel{\text{def}}{=} c_2 - (\pm x_{i_1} \pm x_{j_2}), \quad l_2(x) \leq 0, \\ &\ldots \quad \ldots \quad \ldots \\ l_m(x) &\stackrel{\text{def}}{=} c_m - (\pm x_{i_m} \pm x_{j_m}), \quad l_m(x) \leq 0. \end{aligned}$$

For any minimum perturbation problem $\mathcal{P}$ that has a feasible solution, there exist such (large enough) number $M_{\mathcal{P}} > 0$ that minimizing $f(x)$ under the constraints $\{l_1(x) \leq 0, \ldots, l_m(x) \leq 0\}$ is equivalent to minimizing a continuous piecewise-linear convex function $g(x) \stackrel{\text{def}}{=} 2f(x) + \text{Pos}(Ml_1(x)) + \ldots + \text{Pos}(Ml_m(x))$ under no constraints. Here $\text{Pos}(t) \stackrel{\text{def}}{=} \max(0, t)$ is the "positive part" function; the coefficient "2" in front of "$f(x)$" is needed only for convenience of notation. We do not need to compute M explicitely, it is enough to use the comparison rule

$\alpha_1 M + \beta_1 > \alpha_2 M + \beta_2 \leftrightarrow (\alpha_1 > \alpha_2)$ or $(\alpha_1 = \alpha_2$ and $\beta_1 > \beta_2)$.

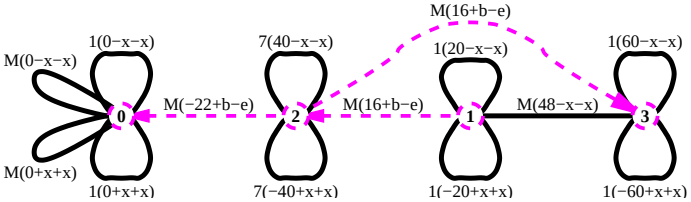
$|t| = \text{Pos}(t) + \text{Pos}(-t)$, so $g(x)$ can be completely represented as a sum of positive parts of linear functions:

$$g(x) = \sum_{i=1}^{n} \Big(\text{Pos}(2a_i(x_i - b_i)) + \text{Pos}(2a_i(b_i - x_i))\Big) + \sum_{i=1}^{m} \text{Pos}(Ml_i(x)). \quad (1)$$

Our graph (denote it G) represents the problem "minimize $g(x)$ under no constraints". Each edge of G represents one item in (1). All the break hyperplanes of $g(x)$ are represented by the edges of G.

EXAMPLE 3.1. *Minimize* $|x_0| + |x_1 - 10| + 7|x_2 - 20| + |x_3 - 30|$ *under the constraints:* $2x_0 \geq 0$, $-2x_0 \geq 0$, $x_2 - x_1 \geq 16$, $x_3 + x_1 \geq 48$, $x_0 - x_2 \geq -22$, $x_3 - x_2 \geq 16$.

The graph representing this problem is shown below.



It remains to apply Simplex Method to the unconstrained piecewise-linear convex minimization problem represented by our graph. If a point of minimum of $g(x)$ satisfies the constraints then it is an optimal rational solution to $\mathcal{P}$. If some point of minimum of $g(x)$ does not satisfy the constraints then $\mathcal{P}$ has no feasible rational solutions. (Note that since $g(x) \geq 0$, $g(x)$ always has a point of minimum).

Call a set of n break hyperplanes which normal vectors are linearly independent a *basic set*. By Lemma 2.2 a subgraph B of G represents a basic set if and only if each connected component of B has exactly one elementary cycle and each such cycle has an odd number of black edges[4]. A point x is said to be *basic* if it is a point of intersection for some basic set of hyperplanes (different basic sets can define the same basic point). $g(x)$ always has a point of minimum, so by the fundamental theorem of linear programming (see [10], p. 19) $g(x)$ always has a basic point of minimum.

Let B be a basic set, and $e \in B$ be some hyperplane in this set. The process of removing e from B and substituting it by some other hyperplane e' so that $(B \backslash e) \cup e'$ is a new basic set is called *a pivot*.

The general simplex method can be briefly described as follows (see [10] pp. 30-84 for details): We start at some *initial basic set* and keep doing pivots according to some *pivoting strategy* until *the stopping criterion* is satisfied. The last visited basic point is a minimum of $g(x)$.

ASSUMPTION 3.2. *(The non-degeneracy assumption) We can always assume that no $n+1$ break hyperplanes intersect at one point, or in other words, that each basic point is defined by exactly one basic set.*

PROOF: Any minimum perturbation problem (or even more generally, any linear convex minimization problem) with a degeneracy can be reduced to a non-degenerate problem by the *small perturbation method* (see [10] p. 78). This method does not require knowing in advance whether or not the problem has a degeneracy. Also it does not require any extra computation until we actually hit a degenerate basic point during the pivoting process. The amount of computation needed to resolve an order-k degeneracy does not exceed the amount of computation needed to make k pivots. □

It remains to specify *the initial basic set*, *the pivoting strategy* and *the stopping criterion* for the graph based simplex method.

- *The initial basic set*:

Since we minimize $g(x)$ under no constraints, any basic point is a feasible basic point. So it is enough to pick any set of n break hyperplanes with linearly independent normal vectors. If such set does not exists we always can add n "remote" hyperplanes defined by $2x_1 \geq -W, 2x_2 \geq -W, \ldots, 2x_n \geq -W$, where W is some very large number.

- *The pivoting strategy*:

We use the greedy pivoting strategy. Let x_{old} be the basic point defined by the basic set $B = \{e_1, e_2, \ldots, e_n\}$.

find_the_outgoing_edge:
for each $e_i \in \{e_1, e_2, \ldots, e_n\}$ **do** {
Consider line L_i defined by $B \backslash e_i$. $x_{old} \in L_i$.
Compute the derivative of $g(x)$ at x_{old} along L_i in both directions.
/* This can be done in amort. $O((m+n)/n)$ time, see Sect. 5. */
if x_{old} is a local minimum of $g(x)$ on L_i **continue**;
else {
e_i is the edge to leave the old basic set. Remember it: $e \stackrel{\text{def}}{=} e_i$;
$L \stackrel{\text{def}}{=} L_i$; $d \stackrel{\text{def}}{=}$ [a direction along L in which $g(x)$ strictly decreases];
break the cycle and **goto** *find_the_incoming_edge*;
}
}
return ("Can not pivot. STOP.");

find_the_incoming_edge:
Move along the line L in the direction d to the next break hyperplane e'.
– *or, saying the same in more detail* –
Parameterize the line L with t.
The labels on all edges become linear functions of t.
Iterate through all edges e'_j of the graph which complement $B \backslash e$ to a basic set, i.e. which have a non-zero coef. at t.
Find the edge (i.e. the break hyperplane) e' that intersects L in the (basic) point x_{new} closest to x_{old} in the direction d.
/* For this we must solve $O(m+n)$ linear equations in t and find the closest to 0 in the direction d root. */

Pivot to the new basic point x_{new} defined by $(B \backslash e) \cup e'$. See Section 5 for a detailed example of using this strategy.

- *The stopping criterion*:

Stop at the basic point x_{old} and output it as a minimum of $g(x)$ if we can not make a pivot according to the pivoting strategy above.

[4]We will use this statement in Section 5.

4. FINDING A $\frac{1}{2}$-NEAR-OPT. SOLUTION

Call $t \in \mathbf{R}$ a semi-integer number if $t - \lfloor t \rfloor = 1/2$.

LEMMA 4.1. *All coordinates of a basic point are either integer or semi-integer. (We omit the proof of this Lemma.)*

Recall Theorem 1.1 from Section 1.

Theorem 1.1 (reformulated) *Let the convex polyhedron formed by the feasible rational solutions of an integer minimum perturbation problem $\mathcal{P}$ have a non-zero volume. Then for any basic optimal rational solution x_{opt} there exists an (integer) $\frac{1}{2}$-near-optimal solution.*

PROOF: By Lemma 4.1 the coordinates of x_{opt} are either integer or semi-integer. Denote the integer coordinates by $\nu_1, \nu_2, \ldots, \nu_k$ and the semi-integer coordinates by $\xi_1, \xi_2, \ldots, \xi_l$. After some permutation of the coordinates $x_{opt} = (\nu_1, \nu_2, \ldots, \nu_k, \xi_1, \xi_2, \ldots, \xi_l)$. We must prove that there exists such point $x_{int} = (\nu_1, \nu_2, \ldots, \nu_k, z_1, z_2, \ldots, z_l)$ that $z_1 = \xi_1 \pm 1/2,\ z_2 = \xi_2 \pm 1/2, \ldots,\ z_l = \xi_l \pm 1/2$ (2) and x_{int} satisfies all the constraints of $\mathcal{P}$. We use induction on l.

The base step: For $l = 2$ the statement of the theorem is an obvious two-dimensional geometry statement.

The inductive step: The first k coordinates of x_{int} are fixed. Substitute their values to the constraints of $\mathcal{P}$. The constraints of $\mathcal{P}$ become constraints on $z_1, z_2, \ldots, z_l$ (those constraints of $\mathcal{P}$ which have two "ν" disappear, those which have one "ν" become one variable constraints, just multiply them by two to bring to the Sum-and-Difference form):

$$\begin{aligned} &s_{11}z_{i_1} + s_{12}z_{j_1} \geq u_1, \\ &s_{21}z_{i_2} + s_{22}z_{j_2} \geq u_2, \quad \text{here } s_{11}, \ldots, s_{p1},\ s_{12}, \ldots, s_{p2} \\ &\quad \ldots \quad \ldots \quad \ldots \qquad\qquad \text{are either 1 or } -1. \\ &s_{p1}z_{i_p} + s_{p2}z_{j_p} \geq u_p, \end{aligned} \tag{3}$$

These constraints still have Sum-and-Difference form and point $(\xi_1, \xi_2, \ldots, \xi_l)$ satisfies them. The set of the possible "$\frac{1}{2}$-near" to $(\xi_1, \xi_2, \ldots, \xi_l)$ integer values for $(z_1, z_2, \ldots, z_l)$ consists of the vertices of the l-dimensional axis-parallel unit cube C centered at $(\xi_1, \xi_2, \ldots, \xi_l)$. For each inequality $s_{*1}z_{i_*} + s_{*2}z_{j_*} \geq u_*$

- either $s_{*1}\xi_{i_*} + s_{*2}\xi_{j_*} = u_*$, i.e. the break hyperplane passes through the center of C
- or $s_{*1}\xi_{i_*} + s_{*2}\xi_{j_*} \geq u_* + 1$, i.e. all the vertices of C belong to the feasible half-space.

Thus the intersection D of cube C with the feasible polyhedron is an intersection of C and a number of half-spaces passing through the center of C. Since the feasible polyhedron is convex and has a non-zero volume, the l-dimensional volume of D is also non-zero. So, intersection of D with one of the hyperfaces F of cube C has a non-zero $l-1$-dimensional volume. Without loss of generality assume that F is defined by equation $z_1 = \xi_1 + 1/2$. The system of inequalities (3), being restricted to F, remains a Sum-and-Difference type system of $z_2, z_3, \ldots, z_l$. Note that any Sum-and-Difference type half-space which break hyperplane passes through the center of C and which intersection with F has a non-zero $l-1$-dimensional volume contains the center of F. So, the center of face F is a feasible point for the system (3) restricted to F. By the induction hypothesis one of the vertices v of F satisfies (3). Thus $x_{int} = (\nu_1, \nu_2, \ldots, \nu_k, < v >)$ is a $\frac{1}{2}$-near-opt. solution. □

Theorem 1.1 shows that a $\frac{1}{2}$-near-optimal solution exists in almost all cases when there exists an optimal rational solution. The only exception are the cases when the rational feasible space is degenerate. This exception is inevitable: fix one coordinate of the rational feasible space to a non-integer number, say $1 \geq 2x_1 \geq 1$. The rational feasible space is non-empty and even $n-1$ - dimensional, but it does not contain any integer point.

Note that the proof of Theorem 1.1 substantially uses the special form of the constraints. The theorem does not hold for the general integer linear programming problem.

Next we prove Theorem 1.2 and show how to find a $\frac{1}{2}$-near-opt. solution. We need to find a combination of signs in (2) such that $(z_1, \ldots, z_l)$ satisfies (3). Denote statement "$z_i = \xi_i + 1/2$" by w_i. Then $\neg w_i$ is the statement "$z_i = \xi_i - 1/2$". For each inequality $s_{*1}z_{i_*} + s_{*2}z_{j_*} \geq u_*$ of (3)

- either $s_{*1}\xi_{i_*} + s_{*2}\xi_{j_*} = u_*$; in that case the inequality is equivalent to the disjunction of two statements: $s_{*1}w_{i_*} \vee s_{*2}w_{j_*}$. Call such an inequality *tight*. Here "$1 \cdot w_*$" denotes the statement "w_*" and "$(-1) \cdot w_*$" denotes the statement "$\neg w_*$".
- or $s_{*1}\xi_{i_*} + s_{*2}\xi_{j_*} \geq u_* + 1$; in that case the inequality is true for any values of $w_1, \ldots, w_l$.

Thus system (3) is equivalent to 2-SAT problem (4).

$$\big(s_{*1}w_{i_*} \vee s_{*2}w_{j_*}\big) \wedge \big(s_{*1}w_{i_*} \vee s_{*2}w_{j_*}\big) \wedge \ldots \wedge \big(s_{*1}w_{i_*} \vee s_{*2}w_{j_*}\big). \tag{4}$$

Each disjunction cluster corresponds to one *tight* inequality of (3). We omit complex subscripts and just put "*" instead of them.

In other words there exists a $\frac{1}{2}$-near-optimal solution x_{int} if and only if there exists a truth assignment for $w_1, \ldots, w_l$ which satisfies (4). If such a truth assignment exists, it defines the coordinates of x_{int}. **Theorem 1.2 is proved**. It remains to solve the 2-SAT problem (4). This can be done in linear time of the number of disjunction clusters in (4) which is $O(m+n)$. A linear time algorithm for 2-SAT is described, for example, in [1].

5. THE TIME COMPLEXITY

In this section we prove that the time complexity of our algorithm is $O\big(p(m+n)\big)$, where p is the number of pivots in Step 1 of the algorithm. As it was shown in Section 4, Step 2 of the algorithm requires only $O(m+n)$ time. So, it remains to prove that the pivoting strategy, given at the end of Section 3, takes $O(m+n)$ time per pivot.

The "bottleneck" of the pivoting strategy is the computation of the derivatives of $g(x)$ in both directions along each of the n lines $L_1, \ldots, L_n$, defined by $B \backslash e_1, \ldots, B \backslash e_n$. The computation of each such pair of derivatives separately requires $O(m+n)$ time. However, next we show that in fact the computation of all the n pairs of derivatives requires just $O(m+n)$ time too. In other words, the amortized cost of one execution of the body of the cycle in the pivoting strategy is $O((m+n)/n)$. Thus we establish the upper bound of $O\big(p(m+n)\big)$ on the time complexity of our algorithm. In parallel we provide a detailed example of doing a pivot according to our pivoting strategy.

Consider the basic point x_{old} defined by the basic set $B = \{e_1, e_2, \ldots, e_n\}$. Without loss of generality we assume that the subgraph $B \subset G$ has just one connected component. According to Lemma 2.2 it consists of an elementary cycle with an odd number of black edges and a number of trees, rooted at the vertices of the cycle. See example on Figure 4.

Rewrite "*find_the_outgoing_edge*" part of the pivoting strategy in a more specific manner:

for each edge, starting at the leafs of the trees, moving towards the cycle and finally going in some fixed direction[5] through the edges of the cycle **do** {

Remove the equation corresponding to the current edge (note that it represents one of the n break hyperplanes which define the old basic point).

This creates a line passing through the old basic point.

This line conveniently parameterizes like it is shown below:

For the current edge we have two possible directions of

[5]Call this direction *positive*. On all figures in this section the positive direction is counterclockwise.

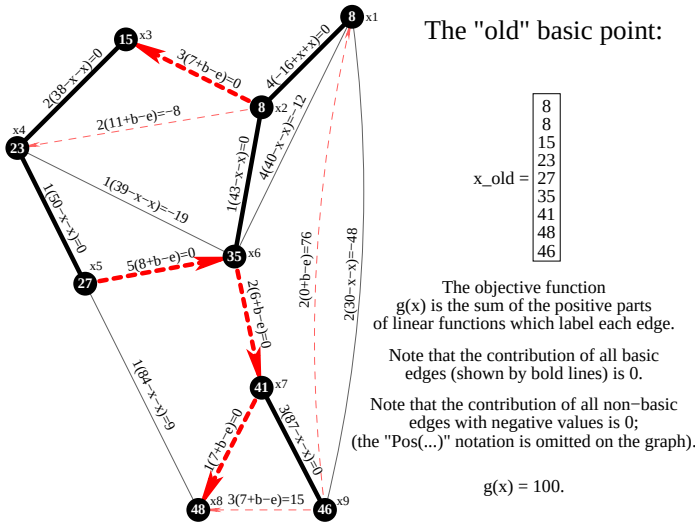


Figure 4: This graph represents the problem: minimize $g(x)=\mathrm{Pos}\big(4(-16+x_1+x_2)\big)+\ldots+\mathrm{Pos}\big(3(7+x_9-x_8)\big)$ under no constraints; each item corresponds to one edge of the graph. The current basic feasible solution is the point x_{old} defined by the system of linear equations corresponding to the bold edges. As before, "red" edges are shown by gray dashed lines.

pivot: $t+0$ and $t-0$.
The objective function $g(x)$ becomes $g(t)$ (for the current edge).
To simplify notation denote $\frac{dg(t)}{d(t-0)}$ and $\frac{dg(t)}{d(t+0)}$ by $d-$ and $d+$ respectively.
Compute $d-$ and $d+$. Depending on the type of the current edge, do this in one of the three different ways described later in this section.
Exactly one of the following three cases takes place: {
case: $d+ < 0$ direction $t+0$ only is beneficial;
case: $d- > 0$ direction $t-0$ only is beneficial;
case: $d- \le 0 \le d+$ there is no beneficial direction for the current edge;
/* Call a direction "*beneficial*" if $g(x)$ strictly decreases along it. */
}
}

- When walking through the edges of the trees, we only care that when we get to a new edge all its children are already processed. Any tree walking strategy with this property can be used.
- After all the tree edges are processed, we pick an arbitrary edge of the cycle (say, the one that has the least address in the memory, or just a random one) and a direction of moving along the cycle, call it *positive*. Starting with this first edge move in the positive direction.

The three ways of computing $d-$ and $d+$ depending on the type of the current edge are as follows:
i) The current edge is a tree edge.
Use formula (5).

$$d- = c\left(\sum_{\text{all } w_i} q_i h_i - \sum_{\text{all } v_i} s_i\cdot(dv_i-)\right);\quad d+ = (d-)+a. \quad (5)$$

It expresses the $d-$ and $d+$ of the current edge via $d-$ of its children (see Figure 5). The correctness of this formula can be checked by direct computation. The number of items in (5) for one edge is $\le$ the number of edges adjacent to its "leafward" vertex. Thus the total number of operations needed to process all the tree edges of B is bounded from above by $2\cdot$[the number of edges of G], i.e. is $O(m+n)$. Figure 6a illustrates step "i)" of our example.

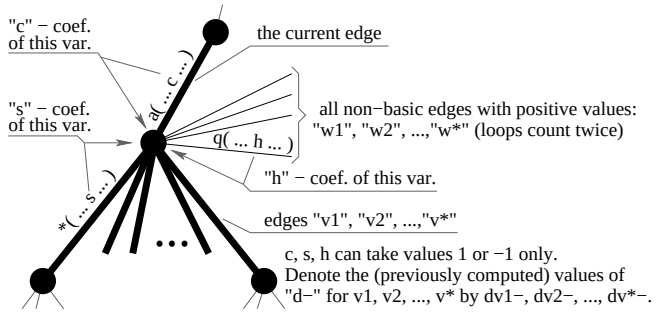


Figure 5: Explanation of notation for Formula (5).

ii) The current edge is the first cycle edge.
In this case we have to spend $O(m+n)$ time for just one edge. Parameterize the first cycle edge with t (see the description of "*find_the_outgoing_edge*" part in this section). The basic edges define a Sum-and-Difference linear system with parameter t. Solve it in respect to t, this takes $O(m+n)$ time. Now the value of each vertex is a linear function of t (see Figure 6b) and $g(L(t))$ has form $\mathrm{Pos}(at)+\overbrace{\mathrm{Pos}(\alpha_1 t+\beta_1)+\mathrm{Pos}(\alpha_2 t+\beta_2)+\ldots+\mathrm{Pos}(\alpha_* t+\beta_*)}^{\text{each item corresponds to one non−basic edge of the graph}}$, where a is the coefficient outside the parenthesis of the first cycle edge.

$$d- = \sum_{i \text{ such that } \beta_i>0} \alpha_i. \qquad d+ = (d-)+a.$$

iii) The current edge is a cycle edge other than the first one.
Use formula (6).

$$d- = c\left(-r\cdot(dz-) + \sum_{\text{all } w_i} q_i h_i - \sum_{\text{all } v_i} s_i\cdot(dv_i-)\right);$$
$$d+ = (d-)+a. \quad (6)$$

It expresses the $d-$ and $d+$ of the current edge via $d-$

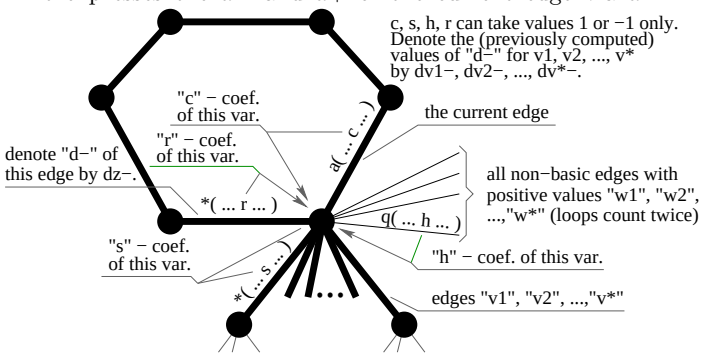


Figure 7: Explanation of notation for Formula (6).

of the basic edges adjacent to its negative-direction vertex (see Figure 7). The correctness of this formula can be checked by direct computation. The number of items in (6) for one edge is $\le$ the number of edges adjacent to its negative-direction vertex. Thus the total number of operations needed to process the cycle edges of B is bounded from above by $2\cdot$[the number of edges of G], i.e. is $O(m+n)$. Figure 6c illustrates step "iii)" of our example.

For each of the groups of edges "i)", "ii)", "iii)" the total runtime is $O(m+n)$. Thus the time compl. of "*find_the_outgoing_ edge*" part of the pivoting strategy is $O(m+n)$ and the time complexity of the whole algorithm is $O\big(p(m+n)\big)$.

To complete our example, execute the remaining part - "*find_the_incoming_edge*" (the $O(m+n)$ runtime upper bound is obvious for this part). In the previous, "*find_the_outgoing_edge*", part any edge with a beneficial direction (i.e. with either $d-<0$ or $d+>0$) can be picked as the outgoing

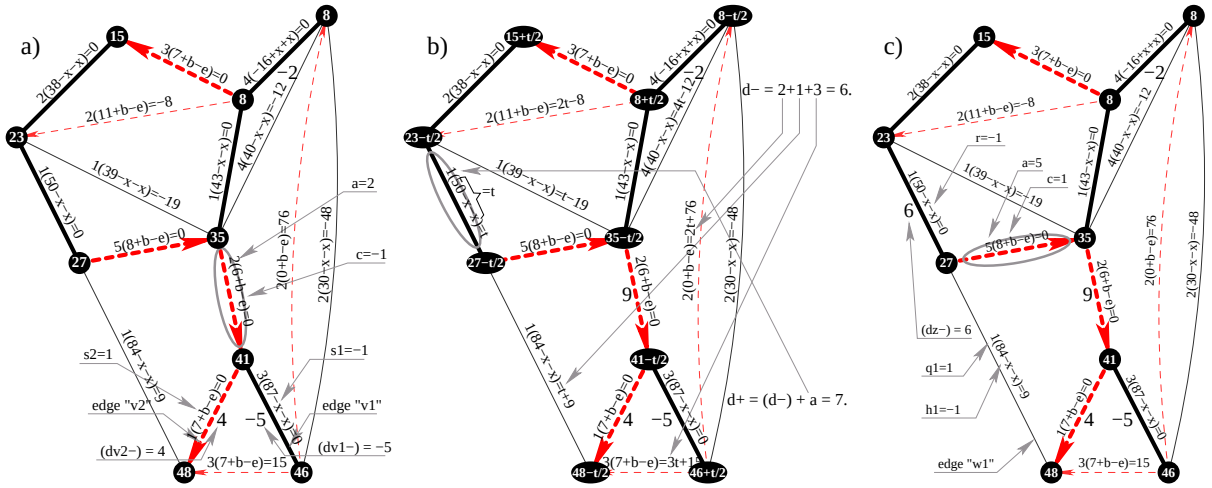


Figure 6: **a)** $d- = c\left(\sum_{\emptyset} q_i h_i - \sum_{i=1}^{2} s_i \cdot (dv_i-)\right) = 9.$ **c)** $d- = c\left(-r \cdot (dz-) + \sum_{i=1}^{1} q_i h_i - \sum_{\emptyset} s_i \cdot (dv_i-)\right) = 5.$

edge. In the greedy strategy we would pick the first edge for which we determined that it has a beneficial direction. But, since in this example we computed $d-$ and $d+$ for all edges, we can choose between several edges; pick the one shown on Figure 8a.

Parameterize the outgoing edge with t (i.e., as before, denote the value of the expression in parenthesis on the edge label by t). $d- = 9$, $d+ = (d-) + a = 11$, $t - 0$ is the beneficial direction. The value of each vertex becomes a linear function of t. Each non-basic edge gives a linear equation of t. Some of them have roots. Actually such an equation has a root if and only if the (candidate to be the incoming) edge that gives this equation satisfies Lemma 2.2. In our example the roots are: $t_1 = -38$, $t_2 = -24$, $t_3 = -9$, $t_4 = -2.5$. The closest to 0 in the beneficial direction (i.e. the greatest negative) root is $t = -2.5$. So, the incoming edge is the one shown on Figure 8b.

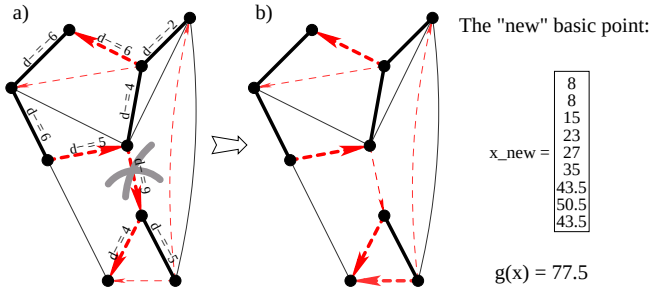


Figure 8: *Find_the_incoming_edge*.

6. CONCLUSION

This algorithm is implemented by the authors in C++. In our implementation we have used a more advanced pivoting strategy than just plain greedy pivots. We pick the steepest gradient direction and follow it until we reach a local minimum of the objective function. The hierarchical constraints are created by a typical $O(nlog(n))$ scanline algorithm from a production layout system. We ran our solver for several hierarchical layout examples and the run time scale very well with the runtime of the constraint creation using a scanline. This makes it a very practical hierarchical solver. The runtime is shown below:

Layout	#Vars	#Constraints	Scanline (s)	Solver (s)
dq0ffs	11030	34140	9.71	8.79
nibble	90157	283682	73.74	178.24
reg64	103231	303399	79.59	111.41

The layout examples are production layouts with artifitial design rule violations introduced. The hierarchical layout *nibble* has 3 levels of hierarchy and 90k shape edges. The resulting layouts are free of violations.

7. REFERENCES

[1] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, March 1979.

[2] R. Bar-Yehuda and D. Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, pages 29:595–609, 2001.

[3] Z. Chen and F. L. Heng. A fast minimum layout perturbation algorithm for electromigration reliability enhancement. In *Proc. of International Symposium on DFT in VLSI Systems*, pages 56–63, 1998.

[4] N. Christofides. *Graph Theory: An Algorithmic Approach*. Academic Press Inc., 111 Fifth Avenue, New York NY 10003, 1978.

[5] F. Heng, L. Liebmann, and J. Lund. Application of automated design migration to alternating phase shifted mask. In *Proc. of ISPD*, pages 38–43, 2001.

[6] F.-L. Heng, Z. Chen, and G. E. Tellez. A vlsi artwork legalization technique based on a new criterion of minimum layout perturbation. In *Proc. of the 1997 International Symposium on Physical Design*, pages 116–121, 1997.

[7] J. Lee and D. Tang. Himalayas - a hierarchical compaction system with a minimized constraint set. In *Proc. of ICCAD*, pages 150–157, 1992.

[8] Y. Z. Liao and C. K. Wong. An algorithm to compact a vlsi symbolic layout with mixed constraints. In *Proc. of DAC*, pages 107–112, 1983.

[9] L. Liebmann and F. Heng. Optimized phase shift migration. US Patent #6083275, July 2000.

[10] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publ. Company, 1984.

[11] D. Marple. A hierarchy preserving hierarchical compactor. In *Proc. of 27th Design Automation Conference*, pages 375–381, 1990.

[12] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[13] L. Y. Wang and Y. T. Lai. Graph theory based simplex algorithm for vlsi layout spacing problems with multiple variable constraints. *IEEE Transactions on CAD*, pages 967–979, August 2001.