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
{llov@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
\[
\pm x_{i1} + x_{j1} \geq c_1, \\
\pm x_{i2} + x_{j2} \geq c_2, \\
\vdots \\
\pm x_{im} + x_{jm} \geq c_m,
\]
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 \( O(p(m + n)) \), where \( p \) is the number of pivots in the simplex algorithm; note that the regular simplex method, being applied to this problem, would require \( O(n^3) \) time.

Our result on production layouts shows that the runtime scale very well with a \( O(n \log(n)) \) 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]: Computer-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 [9]. 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 edge of a 2-variable constraint \( x_i - x_j \geq d \) 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 \( O(nM) \) 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 \( T_n \circ T_{n-1} \cdots \circ T_1 \), 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 transl.) 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 bounded by some distance constraint. Without loss of generality assume that \( A, B \) are vertical and that the constraint has form \( X_B \geq X_A \geq d \). Note that our group of transformations \( G \) is generated by all "+\( k + \beta \)" 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):

\[
\begin{align*}
X_A^* &= T_n \circ T_{n-1} \circ \ldots \circ T_1 (x_A^*) \\
X_B^* &= S_m \circ S_{m-1} \circ \ldots \circ S_1 (x_B^*) \\
X_A^0 &= T_n \circ T_{n-1} \circ \ldots \circ T_1 (x_A^0) \\
X_B^0 &= S_m \circ S_{m-1} \circ \ldots \circ S_1 (y_B^0)
\end{align*}
\]

This gives us 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.
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.

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

Our algorithm consists of two steps. At the first step we find an 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 \( P \) has a non-zero volume then \( P \) has a feasible integer solution \( x_{int} \) such that \( L_\infty \) distance \( \rho_m(x_{opt}, x_{int}) \leq 1/2 \) (call such solution \( \frac{1}{2} \)-near-optimal).

**Proof:** We prove this theorem in Section 4. \( \square \)

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 empty. For example, problem \( f(x) = |x| \), \( 1 \geq x_1 + x_2 \geq 1 \), \( 0 \geq x_1 - x_2 \geq 0 \), \( x_3 + x_3 \geq 1 \), has optimal rational solution \((1, 1, \frac{1}{2})\), 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. \( \square \)

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 exist,
- 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

\[ \sum \text{absolute values of elements in each row of } A \text{ is equal to } 2. \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>24</td>
</tr>
</tbody>
</table>

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. \( \square \)

### 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 \( x_i - x_j \geq c_{ij} \)-type constraint draw a directed red edge from \( x_j \) to \( x_i \) and assign it a (symbolic) weight \( M(\text{"}c_{ij}\" + b - e) \) (see Figure 2).

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

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

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

\(^3\)An elementary cycle is a cycle whose vertices \( x_i, x_{i+1}, \ldots, x_k \) 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 \(a_0, \{x_i - b_i\}\) item in the formula for \(f(x)\) draw two black \(x_i\)-loops and assign them (symbolic) weights \(a_0 ("2b_0 + x - x")\) and \(b_0 ("-2b_0 + x + x")\.

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

\[
\begin{align*}
\lambda_1(x) &= c_1 - (\pm x_i \pm x_j), \\
\lambda_2(x) &= c_2 - (\pm x_i \pm x_j), \\
&\vdots \\
\lambda_m(x) &= c_m - (\pm x_i \pm x_j),
\end{align*}
\]

For any minimum perturbation problem \(P\) that has a feasible solution, there exist such (large enough) number \(M_p > 0\) that minimizing \(f(x)\) under the constraints \(\{\lambda_i(x) \leq 0, \ldots, \lambda_m(x) \leq 0\}\) is equivalent to minimizing a continuous piecewise-linear convex function \(f(x) = \sum_{i=1}^{n} \lambda_i(x)\).

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 \(P\). If some point of minimum of \(g(x)\) does not satisfy the constraints then \(P\) 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 \emph{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 \(c \in B\) be some hyperplane in this set. The process of removing \(c\) from \(B\) and substituting it by some other hyperplane \(c'\) so that \((B \setminus c) \cup c'\) is a new basic set is called a \emph{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 exist we always can add \(n\) "remote" hyperplanes defined by \(2x_i \geq -W, 2x_i \geq -W, \ldots, 2x_i \geq -W\), 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:**
  \begin{itemize}
  \item for each \(e_i \in \{e_1, e_2, \ldots, e_n\}\) do \{
  \begin{itemize}
  \item Consider line \(L_i\) defined by \(B(e_i), x_{old} \in L_i\).
  \end{itemize}
  \}
  \item Compute the derivative of \(g(x)\) at \(x_{old}\) along \(L_i\) in both directions.
  \end{itemize}
  /* This can be done in amort. \(O(m + n)/time\), see Sect. 5. */
  if \(x_{old}\) is a local minimum of \(g(x)\) on \(L_i\), \textbf{continue};

  else \{
  \begin{itemize}
  \item \(e_i\) is the edge to leave the old basic set. Remember it: \(e_i \overset{def}{=} e_j\);
  \item \(L_i \overset{def}{=} L_i; d \overset{def}{=} [a\ direction\ along\ L_i \ \text{in\ which\ } g(x)\ \text{strictly\ decreases}]\);
  \end{itemize}
  break the cycle and \textbf{goto} \textbf{find-the-incoming-edge};
  \}

**find-the-incoming-edge:**
Move along the line \(L_i\) in the direction \(d\) to the next break hyperplane \(e'\).

or, saying the same in more detail –
Parameterize the line \(L_i\) with \(t\).

The labels on all edges become linear functions of \(t\). Iterate through all edges \(e_j\) of the graph which complement \(B(e)\) to a basic set, i.e. which have a non-zero coeff. at \(t\).

Find the edge (i.e. the break hyperplane) \(e'\) that intersects \(L_i\) 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 \((B(e)) \cup e'\). 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. FINDING A \( \frac{1}{2} \)-NEAR-OPT. SOLUTION

Call \( t \in \mathbb{R} \) 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 \( P \) have a non-zero volume. Then for any basic optimal rational solution \( x_{\text{opt}} \) there exists an (integer or semi-) 1/2-near-optimal solution.

Proof: By Lemma 4.1 the coordinates of \( x_{\text{opt}} \) are either integer or semi-integer. Denote the integer coordinates by \( v_1, v_2, \ldots, v_k \) and the semi-integer coordinates by \( \xi_1, \xi_2, \ldots, \xi_l \). After some permutation of the coordinates \( x_{\text{opt}} = (v_1, v_2, \ldots, v_k, \xi_1, \xi_2, \ldots, \xi_l) \). We must prove that there exists such point \( x_{\text{int}} = (v_1, v_2, \ldots, v_k, z_1, z_2, \ldots, z_l) \) such 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 \( x_{\text{int}} \) satisfies all the constraints of \( P \). 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 \( x_{\text{opt}} \) are fixed. Substitute their values to the constraints of \( P \). The constraints of \( P \) become constraints on \( z_1, z_2, \ldots, z_l \) (those constraints of \( P \) which have two "\( \pm \)" disappear, those which have one "\( \pm \)" become one variable constraints, just multiply them by two to bring to the Sum-and-Difference form):

\[
\begin{align*}
\ s_1 z_1 + s_2 z_2 \geq u_1, \\
\ s_2 z_2 + s_3 z_3 \geq u_2, & \quad \text{here } s_1, s_2, s_3, s_4 \geq 0, \quad \text{are either } 0 \text{ or } 1. \\
\ & \quad \text{are either } 0 \text{ or } 1. \\
\end{align*}
\]

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 centered at \((\xi_1, \xi_2, \ldots, \xi_l)\). For each inequality \( s_1 z_1 + s_2 z_2 \geq u \), we have two cases:

- either \( s_1, s_2 z_2 \geq u, \) i.e. the break hyperplane passes through the center of \( C \)
- or \( s_1, s_2 z_2 \geq u, \) 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 hyperplanes \( F \) of cube \( C \) has a non-zero \( l \)-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 \( 2, 3, \ldots, z_l \). 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 \( l \)-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 \( x_{\text{int}} = (v_1, v_2, \ldots, v_k, < v >) \) 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 \( 1 \geq 2 \pi \geq 1 \). The rational feasible space is non-empty and only \( n - 1 \)-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 \( s_1 z_1 + s_2 z_2 \geq u \), we have:

- either \( s_1 \xi_1 + s_2 \xi_2 = u \); in that case the inequality is equivalent to the disjunction of two statements: \( + w \) \( - w \). Call such an inequality tight. Here \( 1 \cdot w_1 \) denotes the statement \( w_1 \) and \( 1 \cdot w_1 \) denotes the statement \( 1 \cdot \neg w_1 \).
- or \( s_1 \xi_1 + s_2 \xi_2 \geq u + 1 \); in that case the inequality is true for any values of \( w_1, \ldots, w_i \).

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

\[
(s_1 w_i \lor s_2 w_i) \land (s_1 w_i \lor \neg s_2 w_i) \land \ldots \land (s_1 w_i \lor \neg s_2 w_i).
\]

Each disjunction cluster corresponds to one tight inequality of (3).

We omit complex subscripts and just put "\( w \)" instead of them.

In other words there exists a \( \frac{1}{2} \)-near-optimal solution \( x_{\text{int}} \) if and only if there exists a truth assignment for \( w_1, \ldots, w_i \) which satisfies (4). If such a truth assignment exists, it defines the coordinates of \( x_{\text{int}} \).

Theorem 1.2 is proved.

In this section we prove that the time complexity of our algorithm is \( O(p(m + n)) \), 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 \( O((m + n)) \). Thus we establish the upper bound of \( O(p(m + n)) \) 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 \( x_{\text{old}} \) defined by the basic set \( B = \{e_1, e_2, \ldots, e_n\} \). Without loss of generality we assume that the subgraph \( B \subset C \) 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 leaves of the trees, moving towards the cycle and finally going in some fixed direction through the edges of the cycle do \( \circ \)

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:

\[ 3(87-x) = 0 \]

For the current edge we have two possible directions of

\( \circ \) Call this direction positive. On all figures in this section the positive direction is counterclockwise.

70
The objective function $g(x)$ is the sum of the positive parts of linear functions which label each edge. Note that the contribution of all basic edges (shown by bold lines) is 0. Note that the contribution of all non-basic edges with negative values is 0. (the "Pos(...)" notation is omitted on the graph).

The current basic feasible solution is the point $x_{\text{old}}$ defined by the system of linear equations corresponding to the bold edges. As before, "red" edges are shown by gray dashed lines.

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

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

i) 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 $\text{Pos}(at) + \text{item corresponds to one non-basic edge of the graph}.$

$$g(x) = 100.$$  

ii) The current edge is the first cycle edge.
We express this formula by $x_{\text{old}}$. 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 \text{[the number of edges of $G$]}$, i.e. is $O(m + n)$. Figure 6a illustrates step "i)" of our example.

iii) The current edge is a cycle edge other than the first one.

Use formula (6).

$$d=\sum \alpha_i$$  

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

\begin{itemize}
  \item 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.
  \item 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.
\end{itemize}

The three ways of computing $d-$ and $d+$ depending on the type of the current edge are as follows:

1) The current edge is a tree edge.

Use formula (5).

$$d= c \left( -r \cdot (dz-) + \sum_{q} q_i h_i - \sum_{s_1} s_1 \cdot (dt_1i) \right) ; \quad d+ = (d-) + a . \quad (5)$$

It expresses the $d-$ and $d+$ of the current edge via $d-$ and $d+$ 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 \text{[the number of edges of $G$]}$, i.e. is $O(m + n)$. Figure 6a illustrates step "i)" of our example.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{Explanation of notation for Formula (5).}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7.png}
\caption{Explanation of notation for Formula (6).}
\end{figure}

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 $O(p(m + n))$.

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) $d− = c \left( \sum q_i h_i - \sum_{i=1}^{\infty} s_i \cdot (dv_i−) \right) = 9$. The layout examples are production layouts with artificial 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


