Objective Variation Simplex Algorithm for Continuous Piecewise Linear Programming

Yu Bai, Zhiming Xu, Xiangming Xi, and Shuning Wang

Abstract: This paper works on a modified simplex algorithm for the local optimization of Continuous PieceWise Linear (CPWL) programming with generalization of hinging hyperplane objective and linear constraints. CPWL programming is popular since it can be equivalently transformed into difference of convex functions programming or concave optimization. Inspired by the concavity of the concave CPWL functions, we propose an Objective Variation Simplex Algorithm (OVSA), which is able to find a local optimum in a reasonable time. Computational results are presented for further insights into the performance of the OVSA compared with two other algorithms on random test problems.

Key words: local optimization; continuous piecewise linear programming; modified simplex algorithm

1 Introduction

Intuitively, the Continuous PieceWise Linear (CPWL) programming technique is useful for continuous optimization. On the one hand, the CPWL functions possess some essential structural properties (e.g., piecewise linearity) which inspire researchers to propose a large number of efficient algorithms [1–7]. On the other hand, any continuous function can be approached by a CPWL function to arbitrary precision [8], so that any continuous nonlinear programming can be solved approximately by CPWL programming.

In the 1960s, Beale et al. [9, 10] applied CPWL approximation and CPWL programming to minimize a separable nonlinear function. Since then, the CPWL programming has attracted a lot of attention and a series of important and efficient techniques and methods have been proposed.

One kind of these methods is to solve the CPWL programming directly with a new version of the simplex algorithm. For example, via expressing one-dimensional CPWL functions by the breakpoints, Fourer [11–13] proposed a general simplex algorithm for minimizing separable CPWL functions. And Fourer and Marsten [14] tested a similar algorithm on XMP subroutine library. Later then, Güder and Nourie et al. [15] presented a dual piecewise-linear simplex algorithm for minimizing convex separable piecewise linear programming. However, this kind of method is only applicable to a special class of CPWL functions, which confines its wide application.

The main idea of another method is to transform the CPWL programming into a series of Linear Programming (LP) problems. Dantzig et al. [16] solved a chemical equilibrium problem by approximating the nonlinear system with a convex separable piecewise model and conducting the transformation of piecewise linear programs to equivalent linear programs. Huang et al. [17] showed a local descent algorithm by solving a series of LP programming in the corresponding linear subregions, but the local optimality criterion can not be
guaranteed in theory. They established a more rigorous local optimality condition for CPWL programming in Ref. [18], based on which a local search algorithm is given. Despite that this kind of methods are suitable for generalized CPWL programming, the implement of equivalent transformations and searching local optimum in the linear subregions will introduce additional variables or constraints which lead to large amounts of calculation and low efficiency.

This paper builds upon the well-known simplex algorithm to develop a general and computationally practical method that searches for a local minimum of an arbitrary CPWL objective, which is linearly constrained. We notice that the global optimization of CPWL programming[19, 20] is getting popular nowadays, but the local optimization algorithms are not efficient enough for such kind of problems in the literature. So it is meaningful to propose a local optimization algorithm with better performance for the generalized CPWL programming.

The rest of this paper is structured as follows. Section 2 discusses the equivalence of the generalized CPWL optimization and the concave piecewise linear optimization. Some relevant preliminaries are also introduced. Section 3 develops the proposed algorithm and discusses the realization issues. Numerical experiments and comparisons with two another methods are conducted in Section 4. Section 5 ends the paper with some conclusions.

2 Method Background

In this section, we will discuss the transformation of the CPWL programming into the CPWL concave programming by exploiting the structure of Generalization of Hinging Hyperplanes (GHH)[21]. To portray the algorithm in a more concise way, some relevant preliminaries will also be introduced.

2.1 Equivalent transformations

A CPWL programming takes the following form:

$$\min f_0(y), \\
\text{s.t. } f_i(y) \leq 0, 1 \leq i \leq m$$

where $y \in \mathbb{R}^n$, $f_i(x)$, $i = 0, 1, 2, ..., m$ are piecewise linear functions.

Huang et al.[18] proved the existence of exact penalty for nonseparable CPWL programming that there exists $\bar{r} > 0$, which makes problem (1) and the following unconstrained CPWL problem (2) possess the identical set of local optimum for any $r \geq \bar{r}$.

$$\min F_r(y) = f_0(y) + r \sum_{i=1}^m \max\{0, f_i(y)\}$$

By utilizing the GHH model, we express $F_r(y)$ as

$$F_r(y) = \sum_{m=1}^M \beta_m \max_{1 \leq j \leq n} \{c_{mn}^T y + d_{mn}\}$$

where $\beta_m \in \{-1, 1\}$, $n_m$ is the number of $c_{mn}^T y + d_{mn}$ functions for $\forall m \in \{1, 2, ..., M\}$. Since GHH model is able to describe any CPWL function.

Clearly, $F_r(y)$ can be rewritten in the following form:

$$F_r(y) = \sum_{m=1}^M \max_{1 \leq j \leq l_m} \{c_{mj}^T y + d_{mj}\} - \sum_{m=1}^M \max_{1 \leq j \leq l_m} \{c_{mj}^T y + d_{mj}\}$$

where all maximal functions are basis functions of the GHH model, $\tilde{M}$ and $\overline{M}$ are the numbers of the positive and negative ones respectively. Obviously, $\min F_r(y)$ is a typical difference of convex functions (D.C.) programming.

Further, by introducing some auxiliary variables $t_m, m = 1, 2, ..., \tilde{M}$, $\min F_r(y)$ is equivalent to the following problem:

$$\min \sum_{m=1}^{\tilde{M}} t_m - \sum_{m=1}^{\overline{M}} \max_{1 \leq j \leq l_m} \{c_{mj}^T y + d_{mj}\},$$

s.t. $c_{mj}^T y + d_{mj} \leq t_m$, $1 \leq j \leq k_m, 1 \leq m \leq \tilde{M}$.

It is not hard to see that the function to be minimized is a concave CPWL function and the constraints are linear. Considering the practical application, we can add bound constraints $l \leq x \leq u$ to make the problem more meaningful.

For simplicity, denote $t = (t_1, t_2, ..., t_M)^T$ and $x = (y^T, t^T)^T$. Then the CPWL programming can be concisely represented as follows

$$\min f(x) = \sum_{i=1}^M \min_{1 \leq j \leq N_i} \{c_{ij}^T x + d_{ij}\},$$

s.t. $Ax \leq b$, $l \leq x \leq u$.

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c_{ij} \in \mathbb{R}^n$, $d_{ij} \in \mathbb{R}$, $\forall 1 \leq i \leq M, 1 \leq j \leq N_i$. Without loss of generality, we assume that problem (6) is nondegenerate, and the feasible domain is denoted by $\Omega = \{x | Ax \leq b, l \leq x \leq u\}$. With the exact penalty theory in Ref. [18] and
a series of equivalent transformations, problem (1) can be transformed into the concave optimization problem (6), which will be focused on in the rest of this paper.

2.2 Preliminaries

In order to exploit the structural properties of a concave CPWL function, some basic concepts should be defined.

Definition 1 Given the GHH function \( f(x) \) in problem (6) and any \( x \in \mathbb{R}^n \), the index sets \( \Psi_i^j \) are defined as follows:
\[
\psi_i^j = \{ j \mid \mathbf{c}_i^j x + d_{ij} = \min_{1 \leq j \leq N_i} \{ \mathbf{c}_i^j x + d_{ij} \} \},
\]
\[\forall 1 \leq i \leq M\]
and the index space \( \Psi_x \) is defined as follows:
\[
\Psi_x = \psi_1^1 \times \psi_2^2 \times \cdots \times \psi_M^M
\]

Definition 2 Given \( \omega = (\omega_1, \omega_2, \ldots, \omega_M) \in \Psi_x \) and \( x \in \Omega \), a facet of \( f(x) \) at the point \( x \) is defined as
\[
\Gamma_x^\omega : \mathbf{c}_x^\omega x + d_x^\omega
\]
where \( c_x^\omega = \sum_{i=1}^M c_{i\omega i} \), \( d_x^\omega = \sum_{i=1}^M d_{i\omega i} \). There are some good and unique properties of the concave piecewise linear functions, such as the concavity and piecewise linearity. To obtain a local optimum, we give a simple and useful property based on the two properties.

Property 1 Given a concave piecewise linear function \( f : \Omega \rightarrow \mathbb{R} \), and \( \Omega_1 \in \Omega \). If \( f(x) = a^T x + b \), \( \forall x \in \Omega_1 \), then we have \( f(x) \leq a^T x + b, \forall x \in \Omega \).

An illustration is given in Fig. 1, and it will be exploited in the next section to derive local optima.

3 Algorithm for Local Minima of CPWL Programming

Our algorithm uses a direct approach based on the traditional simplex algorithm to solve problem (6). It is known that the local optimum of a concave CPWL programming with linear constraints can always be found at some vertices of the feasible set. The geometrical essence of simplex algorithm is to move from one vertex to an adjacent one. It also provides a simple optimality criterion which is easy to implement.

3.1 Bounded-variable simplex algorithm

To set the stage for the proposed OVSA, we take a brief review of the bounded-variable simplex algorithm.

For simplicity, the Bounded-Variable Linear Programming (BVLP) can be expressed as
\[
\min c^T x,
\]
\[\text{s.t. } Ax = b, \quad 0 \leq x \leq u\]
where \( c \in \mathbb{R}^n \), \( x \in \mathbb{R}^n \), \( A \in \mathbb{R}^{m \times n} \), \( b \in \mathbb{R}^m \), \( u \in \mathbb{R}^n \).

To discuss the simplex algorithm for BVLP, we first review the concept of a Basic Feasible Solution (BFS) for BVLP. The decision variables \( x = (x_1, x_2, \cdots, x_n)^T \) can be grouped into two parts which are called basic variables and non-basic variables with the corresponding index sets \( S \) and \( R \).

\( S = \{ i \mid x_i \) is a basic variable\},
\( R = \{ j \mid x_j \) is a non-basic variable\},
and \( S \cup R = I \subset \{1, 2, \ldots, n\} \). Given a solution \( x^0 = (x_1^0, x_2^0, \ldots, x_n^0) \in \Omega \), if there exists an index set \( S = \{i_1, i_2, \ldots, i_m\} \) which satisfies
(1) \( x_i^0 = 0 \) or \( x_i^0 = u_i, \forall j \in I \setminus S \);
(2) the subset of columns of \( A \), denoted by \( P_{i_1}, P_{i_2}, \ldots, P_{i_m} \), are linearly independent;
then \( x^0 = (x_1^0, x_2^0, \ldots, x_n^0) \) is called a basic feasible solution of problem (10). Moreover, \( x_i, i \in S \) are called basic variables, \( B = (P_{i_1}, P_{i_2}, \ldots, P_{i_m}) \) is called a basis matrix. \( x_j, j \in I \setminus S \) are called non-basic variables and the subsets of column vectors of \( A \) indexed by \( I \setminus S \) is called a non-basis matrix denoted by \( N \). Furthermore, we divide the index set \( R \) into \( R_1 = \{ i \in I \setminus S \mid x_i^0 = 0 \} \) and \( R_2 = \{ i \in I \setminus S \mid x_i^0 = u_i \} \), where \( R_1 \cup R_2 = R \).

For given index sets \( (S, R_1, R_2) \), matrix \( A \) can be rearranged as \( A = (B, N_1, N_2) \). We call the triplet \((B, N_1, N_2)\) a basis structure. Similarly, the basic feasible solution \( x \) and the objective coefficients \( c \) can be represented as \( x = (x_B^T, x_{N_1}^T, x_{N_2}^T)^T \), \( c = (c_B^T, c_{N_1}^T, c_{N_2}^T)^T \). Given a basis structure, the corresponding BFS is uniquely obtained as
\[
x_j = 0, \forall i \in R_1
\]
\[
x_i = u_i, \forall i \in R_2
\]
\[
x_B = B^{-1} b - B^{-1} N_1 x_{N_1} - B^{-1} N_2 x_{N_2}
\]
Then the objective of problem (10) is
\[
c^T x = c_B^T x_B + c_{N_1}^T x_{N_1} + c_{N_2}^T x_{N_2} = c_B^T B^{-1} b - (c_B^T B^{-1} N_1 - c_{N_1}^T) x_{N_1} - (c_B^T B^{-1} N_2 - c_{N_2}^T) x_{N_2}
\]
Denote the element in $B^{-1}[N_1, N_2]$ by $\theta_{ij}$, we have
\[ B^{-1}N_1 = (\theta_{ij})_{m \times n_1}, \quad i \in S, \quad j \in R_1, \]
\[ B^{-1}N_2 = (\theta_{ij})_{m \times n_2}, \quad i \in S, \quad j \in R_2, \]
where $n_1$ and $n_2$ are the numbers of the element in $R_1$ and $R_2$, respectively.

Let $a = B^{-1}b = (a_1, a_1, \ldots, a_m)^T$, Eq. (14) is reduced to
\[
c^T x = \sum_{i \in S} c_i a_i - \sum_{j \in R_1} \left( \sum_{i \in S} \left( c_{\theta_{ij}} - c_j \right) x_j \right) - \sum_{j \in R_2} \left( \sum_{i \in S} \left( c_{\theta_{ij}} - c_j \right) x_j \right) = \sum_{i \in S} c_i a_i - \sum_{j \in R_1} \sigma^1_j x_j - \sum_{j \in R_2} \sigma^2_j x_j \quad (15)
\]
where $\sigma^1_j = \sum_{i \in S} c_{\theta_{ij}} - c_j, \quad j \in R_1, \quad \sigma^2_j = \sum_{i \in S} c_{\theta_{ij}} - c_j, \quad j \in R_2$.

Next, we present a local optimality criterion for BFS.

**Lemma 1** (Optimality criterion) A non-degenerate basic feasible solution is optimal for the BVLP if and only if $\sigma^1_j \leq 0, j \in R_1, \quad \sigma^2_j \geq 0, j \in R_2$.

**Proof** Let $x^*$ be a non-degenerate basic feasible solution of problem (10) which satisfies $\sigma^1_j \leq 0, j \in R_1, \quad \sigma^2_j \geq 0, j \in R_2$. Then the objective function equals to
\[
c^T x^* = \sum_{i \in S} c_i a_i - \sum_{j \in R_2} \sigma^2_j u_j.
\]
For any feasible solution $\hat{x}$, the objective equals to
\[
c^T \hat{x} = \sum_{i \in S} c_i a_i - \sum_{j \in R_1} \sigma^1_j x_j - \sum_{j \in R_2} \sigma^2_j x_j \geq \sum_{i \in S} c_i a_i - \sum_{j \in R_2} \sigma^2_j u_j = c^T x^*.
\]
This completes the proof. 

Repeating Ref. [22], we present a method updating BFS to an adjacent start. With an initial BFS $x^0$, the method of BFS update by single-step iteration proceeds as Algorithm 1.

### 3.2 Objective variation simplex algorithm

For a given feasible point $\hat{x}$, suppose there are $k$ facets of $f(x)$ intersecting at $\hat{x}$, which can be expressed as a set of facets,
\[ \Gamma_{\hat{x}} = \{ \Gamma_{\hat{x}}^{\alpha_i} : c_{\hat{x}}^{\alpha_i}^TX + d_{\hat{x}}^{\alpha_i} | \alpha_i \in \Psi_{\hat{x}}, i = 1, 2, \ldots, k \}, \]
where $\Psi_{\hat{x}}$ is the corresponding index space.

Select a candidate $\Gamma_{\hat{x}}^{\alpha_i} \in \Gamma_{\hat{x}}$, then problem (6) can be transformed into an LP as follows:
\[
\min f_i(x) = c_{\hat{x}}^{\alpha_i}^TX + d_{\hat{x}}^{\alpha_i},
\]
s.t. \[ Ax \leq b, \]
\[ l \leq x \leq u \]
In a general situation, for each candidate $\Gamma_{\hat{x}}^{\alpha_i} \in \Gamma_{\hat{x}}$, one should solve LP (16) to obtain an optimum of problem (16), which may not be a local optimum of problem (6). This will take lots of unnecessary computations, especially when the problem scale gets larger. Now we prefer to utilize Algorithm 1 to check the optimality of $\hat{x}$. If $\hat{x}$ meets the optimality criterion with the set of facets $\Gamma_{\hat{x}}$, then we terminate the algorithm and return a local optimum of problem (6). On the contrary, running Algorithm 1 will lead to $\hat{x}$ changed to an adjacent BFS $\check{\hat{x}}$ in quite little time. At the same time, a new set of facets $\Gamma_{\hat{x}}$ can be obtained with $\check{\hat{x}}$. By repeating these procedures mentioned above, we

---

**Algorithm 1** BVS Update by Single-Step Iteration

**Step 1.** Determine the optimality
If $\sigma^1_j \leq 0, \forall j \in R_1, \quad \sigma^2_j \geq 0, \forall j \in R_2$, then the basic feasible solution $x^0$ is optimal. Stop.
Else go to step 2.

**Step 2.** Choose a non-basic variable $x_k$ as an entering variable such that $k = \arg \max((|\sigma^1_k|, |\sigma^2_k|)), k_1 \in R_1, k_2 \in R_2$.

**Step 3.** (Leaving variable rule)

CASE 1: $x_k$ is an entering variable and $\sigma^2_k > 0$.

Compute
\[
\hat{\beta}_{ik} = \min_{\theta_{ik} > 0} \left\{ \frac{x^0_k}{\theta_{ik}} \right\} = \frac{x^0_k}{\theta_{ik}},
\]
\[
s_{rk} = \min_{\theta_{rk} > 0} \left\{ \frac{u_i - x^0_k}{-\theta_{rk}} \right\} = \frac{x^0_k - u_r}{\theta_{rk}},
\]
\[
t_k = \min[u_k, \hat{\beta}_{ik}, s_{rk}] .
\]
(a) If $t_k = u_k$, let $x^1_k = u_k, x^1_i = x^0_i - \theta_{ik} u_k, i \in S, \quad$ Stop.
(b) If $t_k = \hat{\beta}_{ik}$, choose $x_j$ as a leaving variable and let $x^1_i = x^0_i - t_k \theta_{ik}, i \in S, i \neq l, x^1_l = t_k, x^1_l = 0$. Stop.
(c) If $t_k = s_{rk}$, choose $x_r$ as a leaving variable and let $x^1_i = x^0_i - t_k \theta_{ik}, i \in S, i \neq r, x^1_r = t_k, x^1_r = u_r$. Stop.

CASE 2: $x_k$ is an entering variable and $\sigma^2_k < 0$.

Compute
\[
\hat{\beta}_{ik} = \max_{\theta_{ik} < 0} \left\{ \frac{x^0_k}{\theta_{ik}} \right\} = \frac{x^0_k}{\theta_{ik}},
\]
\[
s_{rk} = \max_{\theta_{rk} < 0} \left\{ \frac{u_i - x^0_k}{-\theta_{rk}} \right\} = \frac{x^0_k - u_r}{\theta_{rk}},
\]
\[
t_k = \max\{-u_k, \hat{\beta}_{ik}, s_{rk}\}.
\]
(a) If $t_k = -u_k$, let $x^1_k = 0, x^1_i = x^0_i + \theta_{ik} u_k, i \in S, \quad$ Stop.
(b) If $t_k = \hat{\beta}_{ik}$, choose $x_j$ as a leaving variable and let $x^1_i = x^0_i - t_k \theta_{ik}, i \in S, i \neq l, x^1_l = u_k + t_k, x^1_l = 0$. Stop.
(c) If $t_k = s_{rk}$, choose $x_r$ as a leaving variable and let $x^1_i = x^0_i - t_k \theta_{ik}, i \in S, i \neq r, x^1_r = u_k + t_k, x^1_r = u_r$. Stop.
can obtain a local minimum of problem (6).

In summary, the outline of the objective variation simplex algorithm is given in Algorithm 2.

**Lemma 2** Algorithm 2 for a CPWL programming is a descent algorithm for problem (6) and converges within a finite number of iterations.

**Proof** Given a basic feasible point \( \bar{x} \) and the corresponding set of facets \( \Gamma_x = \{ \Gamma_x^{(i)} : c_x^{(i)} \bar{x}^T + d_x^{(i)} \omega_i, \omega_i \in \Psi_x, i = 1, 2, \ldots, k \} \), select a facet \( \Gamma_x^{(i)} : c_x^{(i)} \bar{x}^T + d_x^{(i)} \omega_i \in \Psi_x \) randomly, we have

\[
f(\bar{x}) = c_x^{(i)} \bar{x}^T + d_x^{(i)}
\]

Take the facet \( \Gamma_x^{(i)} : c_x^{(i)} \bar{x}^T + d_x^{(i)} \omega_i \) as the objective function, and solve the problem (16) with Algorithm 1. A new point \( \bar{x} \) can be obtained with \( \Gamma_x = \{ \Gamma_x^{(i)} : c_x^{(i)} \bar{x}^T + d_x^{(i)} \omega_i \in \Psi_x, i = 1, 2, \ldots, k \} \), for an arbitrary facet \( \Gamma_x^{(i)} : c_x^{(i)} \bar{x}^T + d_x^{(i)} \omega_i \in \Psi_x \), we have

\[
f(\bar{x}) = c_x^{(i)} \bar{x}^T + d_x^{(i)}
\]

Since Algorithm 1 is a natural descent algorithm, so we have

\[
c_x^{(i)} \bar{x}^T + d_x^{(i)} \geq c_x^{(i)} \bar{x}^T + d_x^{(i)}
\]

Considering Property 1, it is clear that

\[
c_x^{(i)} \bar{x}^T + d_x^{(i)} \geq c_x^{(i)} \bar{x}^T + d_x^{(i)}
\]

With Formulas (17)–(20), we have

\[
f(\bar{x}) \leq f(\bar{x})
\]

This lemma reveals that the objective function value is monotonically decreasing during each iteration, and due to finity of the number of vertices of feasible domain \( \Omega \), a local minimum can be found in a finite number of iterations.

### 3.3 Realization issues

The core idea of Algorithm 2 is to change the objective of LP in each iteration. It is plain but efficient. An issue arising from numerical experiments is that the computation time of the set of facets \( \Gamma_x \) accounts for a great proportion in the total computation time of Algorithm 2. To improve the computation efficiency, we propose a computation strategy for \( \Gamma_x \). Given an index space \( \Psi_x = \Psi_x^1 \times \Psi_x^2 \times \cdots \times \Psi_x^M \), let \( N_x^1 \) denote the element numbers of \( \Psi_x^1 \), then the number of combination of the index space \( \Psi_x \) can be calculated as

\[
N_x = N_x^1 \times N_x^2 \times \cdots \times N_x^M
\]

We try to establish a one-to-one correspondence relationship between the number 0, 1, \ldots, \( N_x - 1 \) and the position vector

\[
p_x(i) = (p_{x_1}(i), p_{x_2}(i), \ldots, p_{x_M}(i))^T, i = 1, 2, \ldots, N_x,
\]

which reflect the position of element in \((\Psi_x^1, \Psi_x^2, \ldots, \Psi_x^M)\). For a number \( N \in \{0, 1, \ldots, N_x - 1\} \), dividing \( N \) by \( N_x^1, N_x^2, \ldots, N_x^M \) successively, \( p_x(i) \) can be assigned by the corresponding quotient for each division. For example, let \( M = 3 \) and \( \Psi_x = \{1, 2\}, \Psi_x^2 = \{2, 3, 4\}, \Psi_x^3 = \{5\} \), then we have

\[
N_x^1 = 2, N_x^2 = 3, N_x^3 = 1, \text{and } N_x = 6.
\]

Referring to Fig. 2, we can easily get the position vector

\[
p_x(i) = (p_{x_1}(i), p_{x_2}(i), \ldots, p_{x_M}(i))^T, i = 1, 2, \ldots, 6.
\]

For a \( p_x(i) \), we can get an \( \omega_i \in \Psi_x \) and the corresponding facet \( \Gamma_x^{(i)} \in \Gamma_x \). Then we solve problem (16) with Algorithm 1. Generally speaking, we only need to calculate all possible \( \Gamma_x^{(i)} \in \Gamma_x, i = 1, 2, \ldots, N_x \), when \( x \) is a local optimum while in the other cases, only several \( \Gamma_x^{(i)} \in \Gamma_x \) should be attained to update \( x \). This saves a lot of computation time.

To find an initial basic feasible solution, we can

![Fig. 2 Computational procedure of position index.](image-url)
employ the phase I method or a “big-M” approach (see Refs. [23, 24] for details).

4 Numerical Experiments

In this section, we illustrate the performance of OVSA with randomly generated numerical examples. All random numbers used are generated from a uniform distribution. First, for a concave CPWL programming in the form of Formula (6), we use Matlab simplex to be compared with OVSA. Then we solve DC programming which is piecewise linear using both OVSA and D.C. programming Algorithm (DCA)\textsuperscript{[25]} for comparison. All the experiments are conducted in Matlab R2014a on the platform of Windows7 with AMD Athlon II 3.00 GHz 4.0 GB-RAM.

4.1 Concave CPWL programming

Different from Algorithm 2, we come up with an algorithm based on Matlab simplex. The outline of the algorithm can be described as Algorithm 3.

Matlab simplex is a basic implementation written in Matlab. In Algorithm 3, we obtain the optimum solution of problem (16) with Matlab simplex for in Matlab. In Algorithm 3, we obtain the optimum algorithm can be described as Algorithm 3. The outline of the algorithm based on Matlab simplex. The outline of the algorithm can be described as Algorithm 3.

Algorithm 3 Matlab Simplex Algorithm for Concave CPWL Programming

Initialize
- Find an initial basic solution \( \hat{x} = (x_{B}^T, x_{N_1}^T, x_{N_2}^T)^T \) of problem (6);

Repeat
- Set \( t := 0 \);
- Obtain the set of facets \( \Gamma_{\hat{x}}^o = \{ f_{\hat{x}}^o : c_{\hat{x}}^o^T x + d_{\hat{x}}^o \}_{\hat{x} \in \Psi_k, i = 1, 2, ..., k} \);
- for \( i = 1 \) to \( k \) do
  - Take the facet \( f_{\hat{x}}^o : c_{\hat{x}}^o x + d_{\hat{x}}^o \) into account. Solve the problem (16) with the Matlab simplex and obtain a optimum solution \( \hat{x} \);
  - if \( \hat{x} \neq \hat{x} \) then
    - \( \hat{x} \) is not the local minimum, terminate the for-loop;
    - \( t := t + 1 \);
  - end
- end
- if \( t = = \hat{t} \) then
  - \( \hat{x} \) is the local minimum, terminate the algorithm and return \( \hat{x} \).
- end

Until \( \hat{x} \) is the local minimum

Considering problem (6), the data for the test problems are generated as follows. We consider the cases \([m, n] = [20, 100], [60, 300], [100, 500], [200, 1000], [300, 1500], [400, 2000] \) while \( M = 50, 100, 150, 200, \) separately. Parameters \( c_{ij} \) are located from \([-5, 5] \) while \( d_{ij} \) are located from \([-10, 10] \). \( l_i \) takes value from \([2, 3, 4, 5] \) with equal possibility. \( l_i \) takes value from \([2, 3, 4, 5] \) with equal possibility. \( u_i \) takes value from \([3, 4, 5, 6, 7, 8, 9, 10] \) with equal possibility. \( l_i \) takes value from \([2, 3, 4, 5] \) with equal possibility. \( u_i \) takes value from \([3, 4, 5, 6, 7, 8, 9, 10] \) with equal possibility. \( N_i \) takes value from \([2, 3, 4, 5] \) with equal possibility. \( N_i \) takes value from \([2, 3, 4, 5] \) with equal possibility. \( N_i \) takes value from \([2, 3, 4, 5] \) with equal possibility. \( N_i \) takes value from \([2, 3, 4, 5] \) with equal possibility.

We generate one single starting point for each problem and the same starting points are used in both OVSM and Algorithm 3. For each case of \( m, n \), and \( M \), ten instances are solved. Both of the two algorithms are so-called local algorithms which do not always find the global minimum of the objective functions. Starting from the same point the two algorithms may converge to different local minima. The average and standard deviations of the CPU time and the number of iterations (one-single iteration of simplex algorithm) are recorded in Tables 1–4. As expected, Tables 1–4 clearly demonstrate the superiority of the OVSA on the test problems. The reason for this huge difference on the CPU time and number of iterations is obvious since OVSM is a continuous optimization method and all the useful information is updated based on the former information. However, every linear programming of Matlab simplex algorithm is solved from scratch.

Both of the two algorithms are deterministic algorithms, each of the two algorithms converges to the same local minimum with the same starting point. In addition to the CPU time and the number of iterations, we compare the results of OVSM versus Algorithm 3. In Table 5, columns A and B describe the percentage of problems where OVSM is better than Algorithm 3, Algorithm 3 is better than OVSM, respectively. According to the numerical experiments, we notice that OVSM provides a better result than Algorithm 3 in most instances.

4.2 Piecewise linear DC programming

In Section 2, we declare that a CPWL programming is actually a D.C. programming as follows.

\[
\min \sum_{m=1}^{M} \max_{1 \leq j \leq K_m} \{ c_{mj}^T x + \hat{d}_{mj} \} - \sum_{m=1}^{M} \max_{1 \leq j \leq L_m} \{ c_{mj}^T x + \hat{d}_{mj} \}, \tag{22}
\]

s.t. \( px \leq q \)
where $P \in \mathbb{R}^{m \times n}$, $q \in \mathbb{R}^{m \times 1}$, $\hat{c}_{m,j} \in \mathbb{R}^{n \times 1}$, $\hat{d}_{m,j} \in \mathbb{R}$, $1 \leq m \leq \hat{M}$, $1 \leq j \leq K_m$, $\hat{c}_{m,j} \in \mathbb{R}^{n \times 1}$, $\hat{d}_{m,j} \in \mathbb{R}$, $1 \leq m \leq \hat{M}$, $1 \leq j \leq \hat{L}_m$.

Considering problem (22), in order to assess the performance of OVSM, we will compare with DCA, which is the most popular algorithm for D.C. programming until now.

To illustrate the basic idea of DCA, we introduce the conjugate function of $f(x)$ defined on $\Phi$ as

$$f^*(y) = \sup\{x^T y - f(x) \mid x \in \Phi\}.$$

by Yu Bai et al.: Objective Variation Simplex Algorithm for Continuous Piecewise Linear Programming

### Table 1 Performance of OVSA and Algorithm 3 with $M=50$.

| Problem No. | Problem size | CPU running time (s) | Number of iterations |
|-------------|--------------|----------------------|----------------------|
|             |              | OVSA | Algorithm 3 | OVSA | Algorithm 3 |
|             | $m$ | $n$ | AVG | STD | AVG | STD | AVG | STD |
| 1           | 20 | 100 | 0.24 | 0.03 | 0.90 | 0.25 | 136 | 16 |
| 2           | 60 | 300 | 1.15 | 0.22 | 7.49 | 0.78 | 566 | 80 |
| 3           | 100 | 500 | 3.42 | 0.44 | 22.63 | 7.58 | 1239 | 154 |
| 4           | 200 | 1000 | 14.31 | 0.98 | 131.67 | 32.76 | 2957 | 204 |
| 5           | 300 | 1500 | 48.07 | 3.38 | 388.29 | 194.46 | 5227 | 306 |
| 6           | 400 | 2000 | 107.45 | 4.67 | 932.56 | 314.05 | 7791 | 336 |

### Table 2 Performance of OVSA and Algorithm 3 with $M=100$.

| Problem No. | Problem size | CPU running time (s) | Number of iterations |
|-------------|--------------|----------------------|----------------------|
|             | $m$ | $n$ | OVSA | Algorithm 3 | OVSA | Algorithm 3 |
|             | AVG | STD | AVG | STD | AVG | STD |
| 7           | 20 | 100 | 0.41 | 0.05 | 1.56 | 1.00 | 135 | 19 |
| 8           | 60 | 300 | 2.64 | 0.25 | 8.92 | 1.41 | 716 | 73 |
| 9           | 100 | 500 | 5.54 | 0.58 | 30.16 | 6.87 | 1411 | 145 |
| 10          | 200 | 1000 | 22.74 | 2.56 | 202.03 | 65.07 | 3403 | 382 |
| 11          | 300 | 1500 | 68.75 | 5.53 | 875.84 | 236.22 | 6004 | 488 |
| 12          | 400 | 2000 | 141.19 | 12.10 | 1716.35 | 636.37 | 8738 | 761 |

### Table 3 Performance of OVSA and Algorithm 3 with $M=150$.

| Problem No. | Problem size | CPU running time (s) | Number of iterations |
|-------------|--------------|----------------------|----------------------|
|             | $m$ | $n$ | OVSA | Algorithm 3 | OVSA | Algorithm 3 |
|             | AVG | STD | AVG | STD | AVG | STD |
| 13          | 20 | 100 | 0.58 | 0.08 | 1.35 | 0.30 | 145 | 19 |
| 14          | 60 | 300 | 3.08 | 0.29 | 12.15 | 2.67 | 696 | 66 |
| 15          | 100 | 500 | 8.17 | 0.91 | 33.56 | 8.07 | 1527 | 168 |
| 16          | 200 | 1000 | 31.45 | 3.34 | 276.50 | 54.58 | 3778 | 396 |
| 17          | 300 | 1500 | 87.17 | 9.47 | 886.08 | 269.14 | 6498 | 694 |
| 18          | 400 | 2000 | 171.93 | 13.15 | 2342.09 | 587.89 | 9084 | 611 |

* In this case, one of the instances can not be solved by Algorithm 3 within an acceptable period of time (10 000 s).

### Table 4 Performance of OVSA and Algorithm 3 with $M=200$.

| Problem No. | Problem size | CPU running time (s) | Number of iterations |
|-------------|--------------|----------------------|----------------------|
|             | $m$ | $n$ | OVSA | Algorithm 3 | OVSA | Algorithm 3 |
|             | AVG | STD | AVG | STD | AVG | STD |
| 19          | 20 | 100 | 0.77 | 0.12 | 1.51 | 0.44 | 152 | 22 |
| 20          | 60 | 300 | 4.06 | 0.46 | 13.37 | 5.49 | 746 | 84 |
| 21          | 100 | 500 | 8.17 | 0.91 | 33.56 | 8.07 | 1527 | 168 |
| 22          | 200 | 1000 | 41.18 | 3.25 | 288.73 | 47.16 | 4110 | 319 |
| 23          | 300 | 1500 | 108.48 | 9.47 | 1059.18 | 178.62 | 6962 | 611 |
| 24          | 400 | 2000 | 193.12 | 10.48 | 2530.30* | 510.36 | 9312 | 520 |

* In this case, one of the instances can not be solved by Algorithm 3 within an acceptable period of time (10 000 s).
where $y \in \mathbb{R}^d$. To solve problem (22), two sequences that start from an initial point $x^0$ could be calculated by $y^k \in \partial h(x^k)$ and $x^{k+1} \in \partial g^*(y^k)$ until $x^k$ converge. It is easy to select elements from $\partial h(x^k)$ and $x^{k+1} \in \partial g^*(y^k)$ can be obtained by solving

$$x^{k+1} = \arg \min_{x : P_x \leq q} \{g(x) - (h(x^k) + (x - x^k)^T y^k)\}$$

(23)

which is equivalent to an LP problem for $g(x)$ is a convex CPWL function. The basic procedure of DCA is concluded as Algorithm 4.

The data for the test problems are generated as follows. We consider the cases $[m, n] = [20, 100], [60, 300], [100, 500], [200, 1000]$ while $M = 20$, $\tilde{M} = 20$, and $\tilde{M} = 50, \hat{M} = 50$, separately. The corresponding parameters $\bar{c}_{mn}, \tilde{c}_{mn}, \hat{d}_{mn}, \tilde{d}_{mn}, K_m, L_m, P$, and $q$ are generated in the same way as in Section 4.1.

For each test problem, we run OVSM and DCA to obtain different local minima with the same starting point. We solve ten instances for each case of $m, n, \tilde{M},$ and $\hat{M}$. We use Matlab simplex to solve problem (23) in DCA. The average and standard deviations of CPU time are tabulated in Tables 6 and 7. From Tables 6 and 7, one can see that OVSM takes less CPU time for problem (22). Referring to the standard deviation of CPU times, it is clear that OVSM has greater robustness compared with DCA.

The results of OVSM versus DCA are tabulated in Table 8, columns C and D describe the percentage of problems where OVSM is better than DCA, DCA is

### Table 5: Computational result of OVSM versus Algorithm 3.

| Problem No. | Problem size | Quality of the result (%) |
|-------------|--------------|---------------------------|
|             | $m$ $n$ $M$  | $A$ $B$                    |
| 1           | 20 100 50    | 50 50                     |
| 2           | 60 300 50    | 70 30                     |
| 3           | 100 500 50   | 50 50                     |
| 4           | 200 1000 50  | 70 30                     |
| 5           | 300 1500 50  | 90 10                     |
| 6           | 400 2000 50  | 90 10                     |
| 7           | 20 100 100   | 80 20                     |
| 8           | 60 300 100   | 80 20                     |
| 9           | 100 500 100  | 60 40                     |
| 10          | 200 1000 100 | 80 20                     |
| 11          | 300 1500 100 | 70 30                     |
| 12          | 400 2000 100 | 90 10                     |
| 13          | 20 100 150   | 40 60                     |
| 14          | 60 300 150   | 50 50                     |
| 15          | 100 500 150  | 50 50                     |
| 16          | 200 1000 150 | 40 60                     |
| 17          | 300 1500 150 | 90 10                     |
| 18          | 400 2000 150 | 80 20                     |
| 19          | 20 100 200   | 40 60                     |
| 20          | 60 300 200   | 30 70                     |
| 21          | 100 500 200  | 60 40                     |
| 22          | 200 1000 200 | 80 20                     |
| 23          | 300 1500 200 | 100 0                      |
| 24          | 400 2000 200 | 100 0                      |

Notes: $A$, the percentage of problems where OVSM is better than Algorithm 3; $B$, the percentage of problems where Algorithm 3 is better than OVSM.

### Table 6: Performance of OVSA and DCA with $M=20, \tilde{M}=20$.

| Problem No. | Problem size | CPU running time (s) | OVSA | DCA |
|-------------|--------------|----------------------|------|-----|
|             | $m$ $n$      | AVG      STD      AVG | DCA |
| 1           | 20 100       | 0.41 0.07 | 2.91 4.19 |
| 2           | 60 300       | 1.42 0.14 | 10.75 8.96 |
| 3           | 100 500      | 3.10 0.63 | 17.38 7.54 |
| 4           | 200 1000     | 13.34 1.10 | 71.31 25.10 |

### Table 7: Performance of OVSA and DCA with $M=50, \tilde{M}=50$.

| Problem No. | Problem size | CPU running time (s) | OVSA | DCA |
|-------------|--------------|----------------------|------|-----|
|             | $m$ $n$      | AVG      STD      AVG | DCA |
| 5           | 20 100       | 1.18 0.25 | 18.43 11.52 |
| 6           | 60 300       | 6.34 0.53 | 55.80 20.08 |
| 7           | 100 500      | 12.89 2.39 | 133.93 38.86 |
| 8           | 200 1000     | 40.69 2.62 | 418.36 60.32 |

### Table 8: Computational result of OVSM versus DCA.

| Problem No. | Problem size | Quality of the result (%) | C | D |
|-------------|--------------|---------------------------|---|---|
| 1           | 20 100       | 20 20                     | 60 | 40 |
| 2           | 60 300       | 20 20                     | 90 | 10 |
| 3           | 100 500      | 20 20                     | 70 | 30 |
| 4           | 200 1000     | 20 20                     | 80 | 20 |
| 5           | 20 100       | 50 50                     | 60 | 40 |
| 6           | 60 300       | 50 50                     | 80 | 20 |
| 7           | 100 500      | 50 50                     | 100| 0  |
| 8           | 200 1000     | 50 50                     | 90 | 10 |

Notes: C, the percentage of problems where OVSM is better than DCA; D, the percentage of problems where DCA is better than OVSM.
better than OVSM, respectively. In most cases, Table 8 demonstrates that we can obtain a better result from OVSM than that from DCA.

5 Conclusion

CPWL programming is a widely used model which can be equivalently transformed into a D.C. programming and then a concave CPWL programming. A method with high computational efficiency for the local minimum of the CPWL programming is of great importance and meaning. In this paper, we develop a version of simplex algorithm for generalized CPWL programming. We prove its convergence to a local minimizer and the proposed algorithm is easy to implement. Experimental results are presented which indicate that the algorithm can solve reasonably large size problems in less computational time compared with several existing algorithms.

Acknowledgment

This project was jointly supported by the National Natural Science Foundation of China (Nos. 61473165 and 61134012) and the National Key Basic Research and Development (973) Program of China (No. 2012CB720505).

References

[1] G. Dantzig, S. Johnson, W. White, and M. Science, A linear programming approach to the chemical equilibrium problem, General Information, vol. 5, pp. 38–43, 1958.
[2] S. Zionts, Linear and Integer Programming. Prentice Hall, 1974.
[3] G. B. Dantzig, Linear Programming and Extensions. Princeton, NJ, USA: Princeton University Press, 1963.
[4] A. Charnes and W. W. Cooper, Management models and applications of linear programming, General Information, vol. 4, pp. 38–91, 1957.
[5] A. Charnes and C. E. Lemke, Minimization of non-linear separable convex functionals, Naval Research Logistics Quarterly, vol. 1, pp. 301–312, 1954.
[6] G. B. Dantzig, Recent advances in linear programming, Management Science, vol. 2, pp. 131–144, 1956.
[7] J. K. Ho, Relationships among linear formulations of separable convex piecewise linear programs, Mathematical Programming Study, vol. 24, pp. 126–140, 1985.
[8] L. Breiman, Hinging hyperplanes for regression, classification and function approximation, IEEE Trans. Information Theory, vol. 39, pp. 999–1013, 1993.
[9] E. M. L. Beale, P. J. Coen, and A. D. J. Flowerdew, Separable programming applied to an ore purchasing problem, Applied Statistics, vol. 14, nos. 2&3, pp. 89–101, 1965.
[10] E. M. L. Beale, Numerical methods, in Nonlinear Programming, J. Abadie, ed. Amsterdam, Holland: North-Holland, 1967, pp. 174–177.
[11] R. Fourer, A simplex algorithm for piecewise-linear programming I: Derivation and proof, Mathematical Programming, vol. 33, pp. 204–233, 1985.
[12] R. Fourer, A simplex algorithm for piecewise-linear programming II: Finiteness, feasibility and degeneracy, Mathematical Programming, vol. 41, pp. 281–315, 1988.
[13] R. Fourer, A simplex algorithm for piecewise-linear programming III: Computational analysis and applications, Mathematical Programming, vol. 53, pp. 213–335, 1992.
[14] R. Fourer and R. Marsten, Solving piecewise-linear programs: Experiments with a simplex approach, ORSA Journal on Computing, no. 4, pp. 16–31, 1992.
[15] F. Güder and F. J. Nourie, A dual simplex algorithm for piecewise-linear programming, Journal of the Operational Research Society, vol. 47, no. 4, pp. 583–590, 1996.
[16] G. Dantzig, S. Johnson, and W. White, A linear programming approach to the chemical equilibrium problem, Management Science, vol. 5, no. 1, pp. 38–43, 1958.
[17] X. Huang, J. Xu, and S. Wang, Operation optimization for centrifugal chiller plants using continuous piecewise linear programming, in 2010 IEEE International Conference on Systems Man and Cybernetics (SMC), 2010, pp. 1121–1126.
[18] X. Huang, J. Xu, and S. Wang, Exact penalty and optimality condition for nonseparable continuous piecewise linear programming, Journal of Optimization Theory and Applications, vol. 155, pp. 145–164, 2012.
[19] X. Xi, J. Xu, X. Mu, and S. Wang, Continuous piecewise programming via concave optimization and genetic algorithm, in 2012 IEEE 51st Annual Conference on Decision and Control (CDC), 2012, pp. 2509–2514.
[20] X. Huang, J. Xu, X. Mu, and S. Wang, The hill detouring method for minimizing hinging hyperplanes functions, Computers Operations Research, vol. 39, pp. 1763–1770, 2012.
[21] S. Wang and X. Sun, Generalization of hinging hyperplanes, IEEE Transactions on Information Theory, vol. 51, pp. 4425–4431, 2005.
[22] M. G. Guan and H. D. Zheng, Linear Programming, (in Chinese). Ji’nan, China: Shandong Science &Technology Press, 1983.
[23] J. D. Camm, A. S. Raturi, and S. Tsubakitani, Cutting big M down to size, Interfaces, vol. 25, no. 5, pp. 61–66, 1990.
[24] P. Wolfe, The composite simplex algorithm, SIAM Review, no. 1, pp. 42–54, 1965.
[25] P. D. Tao, The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems, Annals of Operations Research, vol. 133, nos. 1–4, pp. 23–46, 2005.
Yu Bai received the BS degree in 2013 from Beihang University in China. He is currently a PhD candidate at Tsinghua University with his major research interests focusing on continuous piecewise linear optimization and application.

Shuning Wang received the BS degree from Hunan University, China, in 1982, the MS degree and the PhD degree both in the system engineering from Huazhong University of Science and Technology, China, in 1984 and 1998, respectively. He was an associate professor from 1992 to 1993 and a full professor from 1994 to 1995, at the Institute of Systems Engineering, Huazhong University of Science and Technology. He joined Tsinghua University, China, in 1996. Since then, he has been a full professor at Department of Automation, Tsinghua University. His current research interests are mainly in developing practical methods for nonlinear system identification, control and optimization via piecewise linear approximation.

Zhiming Xu received the BS degree in 2006 and MS degree in 2009 from National University of Defense Technology in China. He is currently a PhD candidate at Department of Automation, Tsinghua University, China. He is concurrently a lecturer at the College of Science, Air Force Engineering University, Xi’an, China. He has participated several research projects granted from MOST and NSFC. His active research areas include nonlinear optimization, system engineering, and mobile ad hoc networks.

Xiangming Xi received the BS degree in 2010 from Harbin Institute of Technology in China. He is currently a PhD candidate at Department of Automation, Tsinghua University with his major research interests focusing on continuous piecewise linear optimization modeling and continuous piecewise linear algorithms. His research also covers its applications on support vector machines and portfolio selection.