Joint 48th IEEE Conference on Decision and Control and
28th Chinese Control Conference
Shanghai, P.R. China, December 16-18, 2009

A Fast Geometric Algorithm for Finding the Minimum Distance Between Two Convex Hulls

Dougsoo Kaown and Jianguo Liu

***Abstract*— The problem of computing the minimum distance between two convex hulls has applications to many areas including robotics, computer graphics, path planning, and data classification. In this paper, we propose and investigate a new algorithm (based on the MDM algorithm) for finding the minimum distance between two convex hulls. The new algorithm is simple to understand and easy to implement. The convergence of the algorithm is proved and the performance of the algorithm is compared with the state-of-art techniques. For randomly generated data sets, the new algorithm is more efficient than some of the best available algorithms and implementations, in particular when the number of points and the dimensions are large.**

I. INTRODUCTION

Finding the minimum distance between two convex hulls has important real-world applications. The fields of robotics, animation, computer graphics and path planning all use this distance calculation. In robotics, for example, the distance between two convex hulls is calculated to determine the path of the robot so that it can avoid collisions. Many algorithms designed for minimum distance can also be applied to data classification problems.

The problem of finding the minimum distance between two convex hulls, also known as the nearest point problem (NPP) (see, e.g., Keerthi, Shevade, Bhattacharyya, and Murthy [7]), can be formulated as follows:

Given two sets of points $X = \{x_1, x_2 \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ in $\Re^l$. Let the convex hulls be

$$U = \{ \sum_{i=1}^{m} \alpha_i x_i \ : \ \sum_{i=1}^{m} \alpha_i = 1, \ \alpha_i \geq 0 \ \ (1 \leq i \leq m) \}$$

and

$$V = \{ \sum_{j=1}^{n} \beta_j y_j \ : \ \sum_{j=1}^{n} \beta_j = 1, \ \beta_j \geq 0 \ \ (1 \leq j \leq n) \},$$

respectively. We wish to solve

$$\min \{ \|u - v\| \ : \ u \in U, \ v \in V \} \qquad \text{(NPP)}$$

where $\| \cdot \|$ denotes the general 2-norm in $\Re^l$.

This problem can be reformulated as a closely related minimum norm problem (MNP), i.e., finding the point in a convex hull that is nearest to the origin:

$$\min \{ \|z\| \ : \ z \in Z \}. \qquad \text{(MNP)}$$

Dougsoo Kaown is with the Department of Mathematics, University of North Texas, Denton, TX 75077, USA (dk0008@unt.edu)

Jianguo Liu is with the Department of Mathematics, University of North Texas, Denton, TX 75077, USA (jgliu@unt.edu)

where Z denotes the Minkowski set difference of U and V (Lay [8]):

$$Z = \{ z \ : \ z = u - v, \ u \in U, \ v \in V \}.$$

In fact, every point $z \in Z$ can be expressed as a convex combination of the difference vectors $x_i - y_j$ $(i = 1, 2, \ldots, m, \ j = 1, 2, \ldots, n)$:

$$\begin{aligned} z &= u - v \\ &= \sum_{i=1}^{m} \alpha_i x_i - \sum_{j=1}^{n} \beta_j y_j \\ &= (\sum_{j=1}^{n} \beta_j) \sum_{i=1}^{m} \alpha_i x_i - (\sum_{i=1}^{m} \alpha_i) \sum_{j=1}^{n} \beta_j y_j \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} (\alpha_i \beta_j)(x_i - y_j) \end{aligned}$$

where $\sum_{i=1}^{m} \sum_{j=1}^{n} (\alpha_i \beta_j) = (\sum_{i=1}^{m} \alpha_i)(\sum_{j=1}^{n} \beta_j) = 1$ and $\alpha_i \beta_j \geq 0$. However, we do not want to solve an NPP by solving an equivalent MNP with the difference vectors since the number of difference vectors can be huge (there are mn such vectors).

Many algorithms have been proposed and studied for MNP and for NPP. Gilbert's algorithm [5] is one of the first algorithms for solving MNP. Several years later, Mitchell, Dem'yanov, and Malozemov suggested a new algorithm (the MDM Algorithm [11]). The MDM algorithm works faster than Gilbert's algorithm. The main reason is that Gilbert's algorithm tends to zigzag more when it approaches the solution. Another popular algorithm for solving NPP is the GJK algorithm (Gilbert, Johnson, and Keerthi [6]).

Modifications and improvements have been made on Gilbert's algorithm, the MDM algorithm, and the GJK algorithm, and adaptations have been proposed for other problems such as support vector machines. See, for example, Cameron [1], Chang, Qiao, Wan, and Keane [3], Keerthi et al. [7], and Martin [9].

Our proposed algorithm is based on the MDM algorithm. A novel idea used in the new algorithm is that an approximation $u_k \in U$ to u^* and an approximation $v_k \in V$ to v^* are calculated alternatively at each iteration, where (u^*, v^*) is a solution to (NPP). Convergence is proved and numerical results are reported to compare with some existing methods. For randomly generated data sets, our algorithm is more efficient than some of the best available algorithms and implementations, in particular when the number of points and the dimensions are large.

The rest of the paper is organized as follows. In Section II, we briefly review some exciting algorithms and their modifications and improvements. We introduce the new algorithm and show its convergence in Section III. Numerical results are presented in Section IV, and finally Section V has some conclusion remarks.

II. SOME EXISTING WORK

In this section, we briefly review some existing algorithms and their modifications and improvements. We use $< \cdot, \cdot >$ to denote the common inner product in $\Re^l$. The common 2-norm is defined by

$$\| \cdot \| = \sqrt{< \cdot, \cdot >}.$$

Suppose we wish to find $u^* \in U$ such that

$$\|u^*\| = \min\{ \|u\| \,:\, u = \sum_{i=1}^{m} \alpha_i x_i \,\}.$$

We assume $\|u^*\| > 0$ for MNP and $\|u^* - v^*\| > 0$ for NPP.

A. Gilbert's Algorithm

Gilbert's algorithm was one of the first algorithms for MNP. It locates the point on a convex hull closest to the origin using a piecewise linear path. We may describe the algorithm as follows.

Let P be the convex hull of a set of points in $\Re^l$. Let $z^* \in P$ denote the point with least norm. Define mappings h_P and g_P by

$$h_P(\eta) = \max\{ < \eta, z > :\, z \in P \,\}$$

and

$$g_P(\eta, p) = h_P(\eta) - < \eta, p >, \quad p \in P.$$

Let $s_P(\eta) \in P$ satisfy

$$< s_P(\eta), \eta >= \max\{ < \eta, z > :\, z \in P \,\}.$$

Gilbert's Algorithm for Solving MNP

0) Choose $z \in P$.
1) Compute $h_P(-z)$ and $g_P(-z, z)$. If $g_P(-z, z) = 0$ stop with $z^* = z$; else set $\bar{z} = s_P(-z)$.
2) Compute $\tilde{z}$, the point on the line segment joining z and $\bar{z}$ which has least norm. Set $z = \tilde{z}$ and go to Step 1).

Gilbert showed that z converges to z^* asymptotically if the algorithm does not stop at Step 1) within a finite number of iterations. However, the asymptotic behavior exhibited by the algorithm is slow because of zigzagging.

B. The MDM Algorithm

Several years after Gilbert's work, Mitchell et al. suggested a new algorithm for solving MNP (in 1974) [11]. Their original work was actually published in 1971 in Russian [10]. The MDM algorithm is also simple to describe.

The MDM Algorithm for Solving MNP

0) Initialize $k = 0$ and select $u_0 \in U$.
1) Let the representation of u_k be
$$u_k = \sum_{i=1}^{m} \alpha_i^{(k)} x_i.$$
If u_k satisfies the stopping criterion, stop.
2) Determine $\tilde{x}_{i_k}$ and $\overline{x}_{i_k}$ (among the points x_i) that satisfy
$$< \tilde{x}_{i_k}, u_k >= \max_{\alpha_i^{(k)} > 0} \{< x_i, u_k >\},$$
$$< \overline{x}_{i_k}, u_k >= \min_{1 \le i \le m} \{< x_i, u_k >\}.$$
3) Let $\tilde{\alpha}_{i_k}$ be the coefficient of $\tilde{x}_{i_k}$ in the representation of u_k. Consider the segment
$$\phi_k(s) = u_k + s\, \tilde{\alpha}_{i_k}(\overline{x}_{i_k} - \tilde{x}_{i_k}).$$
Let $s_k \in [0, 1]$ satisfy
$$< \phi_k(s_k),\, \phi_k(s_k) >= \min_{s \in [0,1]} \{< \phi_k(s),\, \phi_k(s) >\}.$$
Update u_k by
$$u_{k+1} = u_k + s_k \tilde{\alpha}_{i_k}(\overline{x}_{i_k} - \tilde{x}_{i_k}).$$
4) Update $k := k + 1$. Go back to Step 1).

The MDM algorithm is similar to Gilbert's algorithm in that both algorithms first choose a descent search direction then find the point with minimum norm on a line segment along that direction. They differ however, on the choice of the search direction. Suppose u_k is the current approximation to the solution. Gilbert's algorithm uses the direction from u_k to one of the x_i that has the least projection on u_k (among all x_i), while the MDM algorithm chooses the direction from one of the x_i that has maximum projection on u_k (among the x_i corresponding to nonzero coefficients in the representation of u_k) to one of the x_i that has the least projection on u_k (among all x_i).

It has been observed that the MDM algorithm converges faster than Gilbert's algorithm. As pointed out in [7], the MDM algorithm tries to crush the total slab toward zero while Gilbert's algorithm only attempts to push the lower slab to zero. Our proposed algorithm is based on the MDM algorithm.

Modifications and improvements have been suggested on Gilbert's algorithm and the MDM algorithms. Adaptations have been made for other problems such as support vector machines (SVM). For example, Keerthi et al. [7] proposed a hybrid algorithm using both Gilbert's idea and the MDM algorithm. Keerthi's algorithm also applies to SVM problems. Purely based on Gilbert's algorithm, Martin [9] suggested a modification for SVM. Both Keerthi and Martin achieved remarkable improvement over the original algorithms, even when applied to SVM problems. Chang et al. made an interesting observation that Gilbert's algorithm would zigzag among the support vectors at the final stage of convergence. They suggested a strategy that improved the performance of Gilbert's algorithm quite significantly for solving MNP.

Another closely related algorithm is the GJK algorithm (Gilbert et al. [6]). It is particularly popular in the field of

robotics. We refer to interested readers the original paper [6] and a later modification by Cameron [1].

We would like to point out that both Gilbert's algorithm and the MDM algorithm can be adapted to solve NPP even though they are designed for MNP, see, e.g., Keerthi et al. [7].

III. THE NEW ALGORITHM FOR NPP AND ITS CONVERGENCE

Let $u^* \in U$ and $v^* \in V$ solve NPP. Then the corresponding difference vector $w^* = u^* - v^*$ will solve the equivalent MNP using the difference vectors $x_i - y_j$ $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$. Let the representations of u^* and v^* be

$$u^* = \sum_{i=1}^{m} \alpha_i^* x_i \quad \text{and} \quad v^* = \sum_{j=1}^{n} \beta_j^* y_j.$$

An important feature of NPP (and MNP) is that for many such problems, most α_i^* and β_j^* are zero in the representation of the solution. The x_i and y_j corresponding to nonzero α_i^* and β_j^* are called support vectors in the support vector machine (SVM) terminology.

One could solve an NPP by forming the difference vectors $x_i - y_j$ $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$ and solving the reformulated MNP. However, this is sensible only if m or n is extremely small, say, $m = 1$ or $n = 1$, since the number of difference vectors is mn which can be prohibitively large. Keerthi et al. [7] showed how to solve an NPP without forming the difference vectors.

One novel idea used in our algorithm is that at each iteration, an approximation $u_k \in U$ to u^* and an approximation $v_k \in V$ to v^* are calculated alternatively, using the difference vectors $x_i - y_j$ only when they are needed. The number of such used pairs will usually be very small comparing with mn. In addition, once u_k has been updated to u_{k+1}, the new approximation u_{k+1} will be used to update v_k. Using newer information often accelerates convergence. Our algorithm can be described as follows.

Our Algorithm for Solving NPP: Algorithm ALT-MDM

0) Initialize $k = 0$ and select $u_0 \in U$ and $v_0 \in V$.

1) Let the representation of u_k and v_k be
$$u_k = \sum_{i=1}^{m} \alpha_i^{(k)} x_i \quad \text{and} \quad v_k = \sum_{j=1}^{n} \beta_j^{(k)} y_j.$$
If u_k and v_k satisfy the stopping criterion, stop.

2) Determine $\tilde{x}_{i_k}$ and $\overline{x}_{i_k}$ (among the points x_i) that satisfy
$$< \tilde{x}_{i_k}, u_k - v_k >= \max_{\alpha_i^{(k)} > 0} \{< x_i, u_k - v_k >\},$$
$$< \overline{x}_{i_k}, u_k - v_k >= \min_{1 \le i \le m} \{< x_i, u_k - v_k >\}.$$

3) Let $\tilde{\alpha}_{i_k}$ be the coefficient of $\tilde{x}_{i_k}$ in the representation of u_k. Consider the segment
$$\phi_k(s) = u_k + s\,\tilde{\alpha}_{i_k}(\overline{x}_{i_k} - \tilde{x}_{i_k}).$$
Let $s_k \in [0, 1]$ satisfy
$$< \phi_k(s_k),\, \phi_k(s_k) >= \min_{s \in [0,1]} \{< \phi_k(s),\, \phi_k(s) >\}.$$
Update u_k by
$$u_{k+1} = u_k + s_k \tilde{\alpha}_{i_k}(\overline{x}_{i_k} - \tilde{x}_{i_k}).$$

4) Determine $\tilde{y}_{j_k}$ and $\overline{y}_{j_k}$ (among the points y_j) that satisfy
$$< \tilde{y}_{j_k}, v_k - u_{k+1} >= \max_{\beta_j^{(k)} > 0} \{< y_j, v_k - u_{k+1} >\},$$
$$< \overline{y}_{j_k}, v_k - u_{k+1} >= \min_{1 \le j \le n} \{< y_j, v_k - u_{k+1} >\}.$$

5) Let $\tilde{\beta}_{j_k}$ be the coefficient of $\tilde{y}_{j_k}$ in the representation of v_k. Consider the segment
$$\psi_k(t) = v_k + t\,\tilde{\beta}_{j_k}(\overline{y}_{j_k} - \tilde{y_{j_k}}).$$
Let $t_k \in [0, 1]$ satisfy
$$< \psi_k(t_k),\, \psi_k(t_k) >= \min_{t \in [0,1]} \{< \psi_k(t),\, \psi_k(t) >\}.$$
Update v_k by
$$v_{k+1} = v_k + t_k \tilde{\beta}_{j_k}(\overline{y}_{j_k} - \tilde{y}_{j_k}).$$

6) Update $k := k + 1$. Go back to Step 1).

Our algorithm updates u_k and v_k alternatively in U and V. The main idea is that when updating u_k to u_{k+1}, the updating strategies used in the MDM algorithm are applied by treating v_k as the origin. Similarly, when updating v_k to v_{k+1}, the same strategies are applied by treating u_{k+1} as the origin.

The main computational cost of the algorithm is on evaluating the inner products to determine $\tilde{x}_{i_k}$, $\overline{x}_{i_k}$, $\tilde{y_{j_k}}$, and $\overline{y}_{j_k}$. When m, n, and l are large which is the case for some application problems, directly evaluating the inner products for every new u_k and v_k can be very costly. Following the idea used in Platt [12] and Keerthi et al. [7], we calculate the inner product $< x_i, y_j >$ only when it appears, and we cache all calculated inner products for later use. The inner product $< x_i, u_k - v_k >$ or $< y_j, v_k - u_{k+1} >$ will never be directly evaluated. Instead, these inner products will be evaluated by using the representations of u_k, u_{k+1}, and v_k in x_i and y_j and the inner products between x_i and y_j.

We now turn to the convergence of the algorithm. The proofs follow quite nicely with the line in Mitchell et al. [11]. We omit the details of the proofs for most lemmas and theorems here because of their length. Interested readers may find the detailed proofs in [4].

The following result can be seen immediately.

Lemma 1: $\|u_{k+1} - v_{k+1}\| \le \|u_{k+1} - v_k\| \le \|u_k - v_k\|$.

Following Mitchell et al. [11], we first define some notations. For the sake of convenience, we may omit the iteration index k when there is no confusion. Let $u \in U$ and $v \in V$. Define

$$\delta_x(u, v) =< u, u - v > - \min_i < x_i, u - v >, \qquad (1)$$

$$\delta_y(u, v) =< v, v - u > - \min_j < y_j, v - u >, \qquad (2)$$

$$\delta(u, v) = \delta_x < u, v > + \delta_y < u, v >, \qquad (3)$$

$$\Delta_x(u,v) = \max_{\alpha_i>0} < x_i, u-v > - \min_i < x_i, u-v >, \quad (4)$$

$$\Delta_y(u,v) = \max_{\beta_j>0} < y_j, v-u > - \min_j < y_j, v-u >, \quad (5)$$

$$\Delta(u,v) = \Delta_x(u,v) + \Delta_y(u,v). \quad (6)$$

Geometrically, $\delta_x(u,v)$ is the difference between the projection of u on $u-v$ and the smallest projection of x_i on $u-v$, while $\Delta_x(u,v)$ is the difference between the largest projection of the x_i corresponding to nonzero coefficients in the representation of u on $u-v$ and the smallest projection of x_i on $u-v$. Similar interpretations apply to $\delta_y(u,v)$ and $\Delta_y(u,v)$. Since u is a convex combination of x_i and v is a convex combination of y_j, we have

Lemma 2:

$$\Delta_x(u,v) \geq \delta_x(u,v) \geq 0,$$

$$\Delta_y(u,v) \geq \delta_y(u,v) \geq 0.$$

The first theorem gives a necessary and sufficient condition for $u \in U$ and $v \in V$ to be a solution of NPP.

Theorem 1: A pair $u \in U$ and $v \in V$ solves NPP iff $\Delta(u,v) = \delta(u,v) = 0$.

Proof: For a detailed proof, please see [4]. We can intuitively see that (u,v) is a solution to NPP iff the smallest projection of x_i on $u-v$ is the same as the projection of u on $u-v$, and the smallest projection of y_j on $v-u$ is the same as the projection of v on $v-u$. In other words, iff $\delta(u,v) = 0$. Similar arguments can be made for $\Delta(u,v)$. ■

The next two lemmas will be used for proving a major convergence theorem.

Lemma 3: The step lengths s_k and t_k used in Step 3 and Step 5 can be expressed as

$$s_k = \min\{1, \frac{\Delta_x(u_k, v_k)}{\tilde{\alpha}_{i_k} \left\| \overline{x}_{i_k} - \tilde{x}_{i_k} \right\|^2}\},$$

$$t_k = \min\{1, \frac{\Delta_y(u_{k+1}, v_k)}{\tilde{\beta}_{j_k} \left\| \overline{y}_{j_k} - \tilde{y}_{j_k} \right\|^2}\}.$$

Proof: This can be seen by direct computation, using the fact that both the inner products $< \phi_k(s), \phi_k(s) >$ and $< \psi_k(t), \psi_k(t) >$ are convex quadratic functions in s and t, respectively. ■

Lemma 4:

$$\lim_{k\to\infty} (\tilde{\alpha}_{i_k} \Delta_x(u_k, v_k) + \tilde{\beta}_{j_k} \Delta_y(u_k, v_k)) = 0. \quad (7)$$

Proof: Please see [4]. ■

A major convergence theorem is the following.

Theorem 2:

$$\underline{\lim}_{k\to\infty} \delta(u_k, v_k) = 0. \quad (8)$$

Proof: Please see [4]. ■

To establish the convergence theorem, we need one more lemma.

Lemma 5: Suppose $\{u_k \in U,\ v_k \in V\}$ is a sequence that satisfy $\|u_{k+1} - v_{k+1}\| \leq \|u_k - v_k\|$ and there is a subsequence that satisfy $\delta(u_{k_j}, v_{k_j}) \to 0$. Then the sequence $\{u_k - v_k\}$ converges to $u^* - v^*$ where $u^* \in U$ and $v^* \in V$ solve NPP.

Proof: Please see [4]. ■

Now we are ready to state the convergence result. Our algorithm is named Algorithm ALT-MDM.

Theorem 3: The sequence $\{u_k - v_k\}$ generated by Algorithm ALT-MDM converges to $u^* - v^*$ where $u^* \in U$ and $v^* \in V$ solve NPP.

Proof: It follows from Lemma 1, Theorem 2, and Lemma 5. ■

We note that there is no need to show the convergence of $\{u_k\}$ or $\{v_k\}$ since we maintain $u_k \in U$ and $v_k \in V$ for all k. In addition, the solution to NPP is not unique in general, though MNP has a unique solution. Once a stopping criterion is satisfied, $u_k \in U$ and $v_k \in V$ will be an approximate solution to NPP.

The following result is geometrically interesting, in particular in view of support vector machines.

Theorem 4: Let $u^* \in U$ and $v^* \in V$ be a solution to NPP. Let

$$U^* = \{\, u \in U \ :< u, u^* - v^* >=< u^*, u^* - v^* > \}, \quad (9)$$

$$V^* = \{\, v \in V \ :< v, v^* - u^* >=< v^*, v^* - u^* > \}. \quad (10)$$

Then $u_k \in U^*$ and $v_k \in V^*$ for all k sufficiently large.

Proof: Please see [4]. ■

The sets $U^* \subset U$ and $V^* \subset V$ are two facets that are parallel and are the convex hulls generated by all support vectors. Any $u \in U^*$ and $v \in V^*$ will be a solution to NPP if $u-v$ is perpendicular to U^* or V^*. Theorem 4 says $\{u_k\}$ and $\{v_k\}$ will eventually land on U^* and V^*, respectively and stay there.

IV. Numerical Experiments

In this section, we compare our algorithm ALT-MDM with one of the best algorithms, the sequential minimization optimization algorithm (SMO) by Platt [12]. More specifically, we compare with one of the best SMO-type implementation packages, LIBSVM (Version 2.85), by Chang and Lin [2].

We implemented our algorithm in Matlab, running on a Pentium 4 desktop. Our algorithm may run more efficiently if implemented in C or Fortran.

The numerical experiments were conducted on randomly generated data. We use m and n to denote the number of points in X and Y, and use l to denote the dimension of those points. Once m, n, and l are given, we use the Matlab function $rand$ to generate $X = rand(m,l). * exp(rand(m,l))$ and $Y = -rand(n,l). * exp(rand(n,l))$, where Row i of X is x_i $(1 \leq i \leq m)$ and Row j of Y is y_j $(1 \leq j \leq n)$. For each case, five problems were generated and solved by both LIBSVM and ALT-MDM. We report the average CPU time (in seconds) for the five problems.

The test problems are all linearly separable due to the way they are generated. We use LIBSVM with linear kernel and set the parameters $-c$ as 10^{10} and $-e$ as 10^{-10}. We stop ALT-MDM when

$$\Delta(u_k, v_k) \leq 10^{-10}.$$

It turns out that LIBSVM and ALT-MDM always identify the same support vectors for these randomly generated problems, and the difference between the minimum values found by the two algorithms is within 10^{-9}.

We now present a few figures. Again m is the number of points in X, n is the number of points in Y, and l is the dimension. For given m and n, we use four l values: $l = (1/4)n, (1/2)n, (3/4)n, n$.

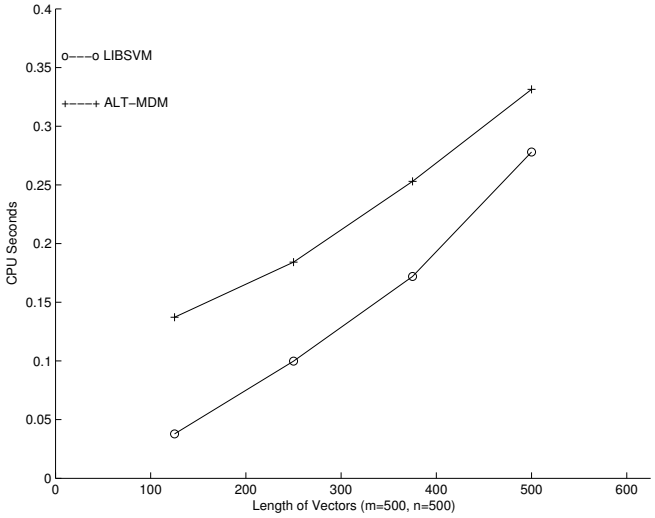

Fig. 1. Compare ALT-MDM with LIBSM: $m = n = 500$

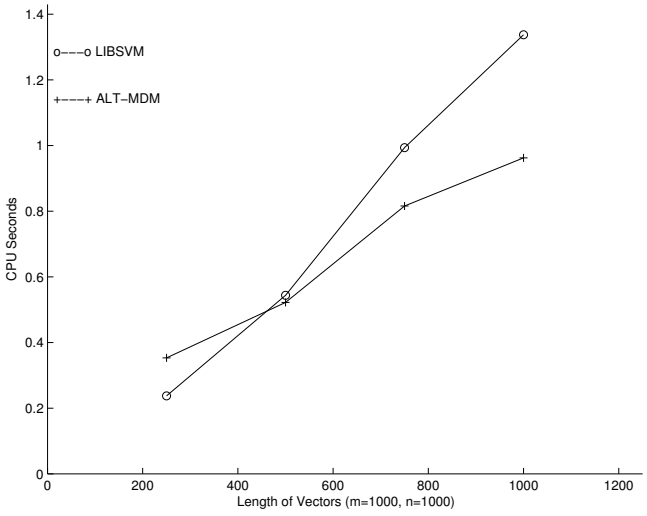

Fig. 2. Compare ALT-MDM with LIBSM: $m = n = 1000$

From Figure 1, we see that LIBSVM outperforms ALT-MDM when m and n are relatively small. However, Figures 2, 3, 4, and 5 show that when m and n are relatively large

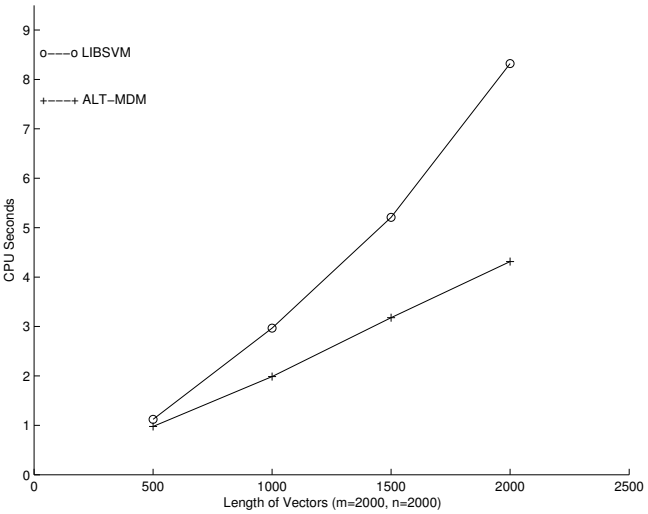

Fig. 3. Compare ALT-MDM with LIBSM: $m = n = 2000$

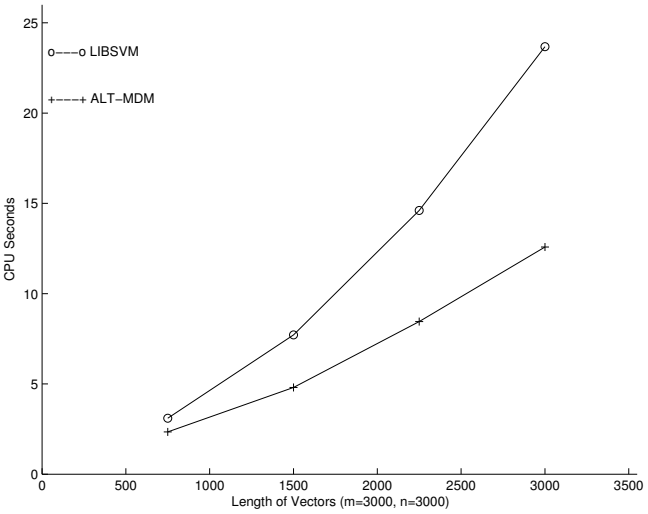

Fig. 4. Compare ALT-MDM with LIBSM: $m = n = 3000$

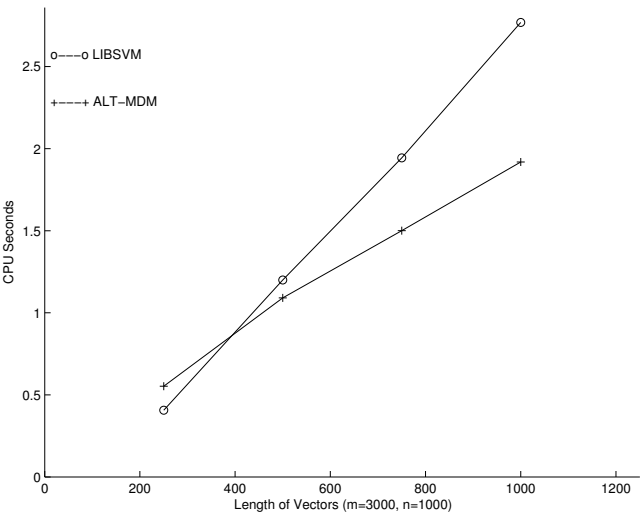

Fig. 5. Compare ALT-MDM with LIBSM: $m = 3000, n = 1000$

$(m, n \geq 1000)$, ALT-MDM outperforms LIBSVM, and the margin grows with the dimension. It indicates that ALT-MDM may have good potential for solving large scale data classification problems.

V. CONCLUSIONS

We have presented a new algorithm based on the MDM algorithm. A novel idea is that the updating strategy is applied alternatively to $u_k \in U$ and $v_k \in V$, and newer information is used once it is available. Numerical experiments show that our algorithm outperforms some of the best available algorithms on randomly generated data, in particular when the size of problems grows. Since such inner-product based algorithms can be easily adapted for SVM problems, the new algorithm may have considerable potential for SVM. We plan to implement the new algorithm with kernel functions and test it using benchmark SVM problems. The result will be presented in a future report.

VI. ACKNOWLEDGMENTS

The authors thank Robert Kallman, John Neuberger, and Xiaohui Yuan for their helpful comments, and thank the referees for many constructive suggestions.

REFERENCES

[1] S. Cameron, "Enhancing GJK: computing minimum and penetration distances between convex polyhedra", *Proceedings of the Int. Conf. Robotics and Automation*, 1997.

[2] C. C. Chang and C.J. Lin, *LIBSVM: A Library for Support Vector Machines*, (software available at $http$: $//www.csie.ntu.edu.tw/\tilde{}cjlin/libsvm$), 2001.

[3] L. Chang, H. Qiao, A. Wan, and J. Keane, "An Improved Gilbert Algorithm with Rapid Convergence", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3861-3866, 2006.

[4] K. Dougsoo and J. Liu, "A Geometric Algorithm for Finding the Minimum Distance Between Two Convex Hulls", manuscript, 2009.

[5] E. G. Gilbert, "Minimizing the quadratic form on a convex set", *SIAM J. Contr.*, vol. 4, pp. 61-79, 1966.

[6] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three dimensional space", *IEEE J. Robot. Automat.*, Vol. 4, pp. 193-203, 1988.

[7] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design", *IEEE Trans. Neural Networks*, vol. 11, pp. 124-136, 2000.

[8] S. R. Lay, *Convex Sets and Their Applications*, New York: Wiley, 1982.

[9] S. Martin, "Training support vector machines using Gilbert's algorithm", *Proceedings of the Fifth IEEE International Conference on Data Mining*, 2005.

[10] B. F. Mitchell, V. F. Dem'yanov, and V. N. Malozemov, "Finding the point of a polyhedron closest to the origin", *Vestinik Leningrad. Gos. Univ.*, Vol. 13, pp. 38-45, 1971 (in Russian).

[11] B. F. Mitchell, V. F. Dem'yanov, and V. N. Malozemov, "Finding the point of a polyhedron closest to the origin", *SIAM J. Contr.*, vol. 12, pp. 19-26, 1974.

[12] J. Platt, "Fast training of support vector machines using sequential minimal optimization", in *Advances in Kernel Methods - Support Vector Learning*, B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors, MIT Press, pp. 185-208, 1999.