Brief Proposal: Quadratic spine similarity approximation (for Unicode glyphs).

Anthony Y. Fu
Email: ayf@mit.edu
Affiliation: Dept. of CS, CityU of Hong Kong, and CSAIL, MIT, USA

How it works

The kernel density estimation (KDE) works well with Unicode symbols' similarity assessment but the as you know. We have to sample at least 100 points from alphabetical list and at 200 points for Chinese characters representation. The similarity list construction for the one hundred readable characters in ASCII took us a week to compute with two PCs.
True type fonts represent characters with quadratic spline, so the algebraic method can be applied. I tried to deduct the accurate math representation, but it is a too large formula. Former mathematicians have worked on the part of the formula approximation. What I need to do is to approximate it to the end. I just found Taylor series expansion could work. I'll try to prove the method works through experiments.

Back Ground Ref:
// -----Start here-----

Safeguard against *Unicode Attack*: UC-SimList Generation using Kernel Density Estimation

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General – *Security and protection.*

General Terms
Security, Legal Aspects, and Verification.

Keywords
Unicode, Phishing, and Secure Web Identity.

1. Abstract

We identify a problem of using Unicode [11] in the Web that could cause a severe security problem. The potential security attack derives from the fact that there are too many similarity characters in the Unicode Character Set (UCS [11]). The foundation of solving

such a problem relies on the solution of evaluating the similarity of characters in UCS. We propose using a renowned Kernel Density (KDE) evaluation solution to establish such a Unicode Similarity List (UC-SimList).

2. Introduction/*Unicode Attack*

With the population of the Internet, people from various countries/regions/cultures participate into the information exchange evolution. Nowadays, we can find most of the spoken/written languages in the world appearing in the Web. The biggest cornerstone of making these characters from different languages possible relies on the utilization of Unicode. UCS is a character repertoire with character codes. The most popular version of UCS is using 16 bits to represent on character code and a lot of visually similar characters coexist the UCS. Figure 1 shows the samples of similar characters that exist in UCS. “b” (U+0062), “a” (U+0061), “n” (U+006E), and “k” (U+006B) are the original English characters in UCS, while the other ones are the similar characters corresponding to the original ones.

b	ｂ	ḅ	ḇ	þ
0062	FF42	1E05	1E07	00FE
a	а	ａ	ạ	ḁ
0061	0430	FF41	1EA1	1E01
n	ｎ	ņ	п	ṇ
006E	FF4E	0146	043F	1E47
k	ｋ	ƙ	ḳ	ḵ
006B	FF4B	0199	1E33	1E35

Figure 1. **Similar characters to “b”, “a”, “n”, and “k” (in Arial Unicode MS Font). Code under each character is the character code in hexadecimal form**

The coexistence of similar characters in UCS could cause a series of Web security problems, such as Spamming, Phishing, Web Identity Faking.

Spamming Attack

On the spamming aspect, malicious people (spammers) can create numerous spams while keeping the appearance of original email as shown in Figure 2.

A	n		a	p	p	l	e		a		d	a	y	,				
41	6E	20	61	70	70	6C	435	20	61	20	64	61	79	2C	20			
k	e	e	p	s		d	o	r	c	t	o	r		a	w	a	y	!
6B	65	65	70	73	20	64	e	72	63	74	6F	72	20	61	77	61	79	21
A	n		a	p	p	l	65		a		d	a	y	,				
410	FF4E	20	430	70	70	49	443	20	FF41	20	64	430	FF59	2C	20			

k	e	e	p	s		d	o	r	c	t	o	r		a	w	a	y	!
FF4B	435	FF45	70	455	20	217E	6F	FF52	63	FF54	FF4F	72	20	61	77	61	79	1C3

Figure 2. Sample of Sapmming. The first sentence is the original one, while the second one is the mutated one. Code under each character is the character code in hexadecimal form without 0-padding in the left side.

Phishing Attack

On the Phishing aspect, malicious people (phishers) could replace visually similar characters into the Internationalized Resource Identifier (IRI) [4] to mimic the real one, as shown in Figure 3.

w	w	w	.	e	b	a	y	.	c	o	m
77	77	77	002E	65	62	61	79	002E	63	006F	006D
w	w	w	·	e	b	a	y	.	c	o	m
77	FF57	FF57	2027	FF45	62	FF41	443	002E	441	03BF	006D
w	w	w	·	e	b	a	y	·	c	o	m
FF57	FF57	77	FF65	FF45	FF42	61	79	00B7	63	03BF	217F

Figure 3. Samples of Phishing. The first weblink is the original/real one and the rest two weblinks are mutated/faked ones. Code under each character is the character code in hexadecimal form.

Another possibility is that the original website/webpage could be replaced by similarity characters, such that no existing Anti-Phishing systems (i.e., [3][5][6][7][8][12][16]) can catch it, because these anti-phishing detector must know there is a sensitive word in the email or webpage before suspect one webpage to avoid burdening the syste too much by checking all of the weblinks. E.g., in the system built by SiteWatcher [5][12][16], the system only suspect a webpage when a sensitive word is found in the webpage.

Web Identity Stealing/Faking/Attack

There are many cases that the Web based systems (Website, Email, Chatting Room, BBS, Instant Message, Blog, Wiki, etc.) utilize text string to represent the user name. There will be no problem if only ASCII [1] characters are permitted to use as user name, however, if the Unicode string is used to represent the user name, the system is possible to be under attack, especially when the user name is the only way for users to identify each other, such that malicious people may successfully gain the possible victims' trust.

3. Related Works

We could see there are many types of *Unicode attacks* and all of them are utilizing the fact of the coexistence of similar characters in the UCS. The possibility of using similar

characters to generate fake domain name using national alphabets is firstly reported in [13] and named *Homograph Attack*. This is a general concept. One of potential standard Internationalized Resource Identifier (IRI) [4] utilizes the UCS as the character set to locate Web resources, and the potential phishing attack is investigated and reported in [14]. It focuses on the survey of UCS which is the most populated character set for the internationalization of Web information, thus we would like to give it a more specific name, *Unicode Attack*.
The basic idea of carrying out *Unicode attack* is to generate the mutations of the original (Unicode) string (such as spam content, domain name, and user name, etc.) by replacing similar characters, and the basic idea of safeguarding the (Web) systems from *Unicode attack* is to evaluate the similarity of the suspected string(s) to the original one(s). A generic methodology for the counter measure against *Unicode attack* has been reported in [15], in which the construction of the UC-SimList is considered as a critical part of the Unicode string similarity evaluation.
Former researches have worked on the constructions of UC-SimList, where the pixel overlapping evaluation method has been used [2]. Although the method is straightforward, it does not perform well when certain amount of departure of glyph contour is consisted in the evaluations cases. In this paper, we propose to use kernel density (KDE) instead.

About UC-SimList

The first UC-SimList construction method is proposed in [14], however, we would like re-address it to make the paper content concrete. UC-SimList is a matrix, from which we can easily find the similarity of a given pare of characters. The construction of UC-SimList requires of two sub-lists, UC-SimList_v and UC-SimList_s.
UC-SimList_v is a matrix, from which we can easily find the visual similarity of a given pare of characters, such as the visual similarity of "a" (U+0061) and "а" (U+0430) is 100% and we denote it as Sim_v("a", "а")=100%. The construction of UC-SimList_v could be considered as a pattern similarity evaluation problem and we construct it in an automatic way.
UC-SimList_s is also a matrix, from which we can easily find whether a given pair of characters is semantically equivalent, such as the semantic similarity of "a" (U+0061) and "A" (U+0041) is 100%, and we denote it as Sim_s("a", "A")=100%. The construction of UC-SimList_s may need much more human intervene. However, the alphabetic lists of different languages could help to solve most of the problems, such as English (Upper/Lower Cases), Chinese (Simplified/Traditional Characters), and Japanese (Katakana/Hiragana). Indubitably, much more languages and language usage customs should be considered and included into the UC-SimList_s.
//Delete?:The construction of UC-SimList needs both UC-SimList_v and UC-SimList_s.//
To construct UC-SimList, we first need to find the similar characters in UC-SimList_s for a given character, e.g. we find two characters, "a" and "A", are semantically similar to "a" (including "a" itself). We use the semantically similar characters as a source and find

all of the visually similar characters of the source from UC-SimList_v, e.g. we find "а" (U+0430), "ａ"(U+FF41), "A"(U+0491), "Ａ"(U+FF21), "А"(U+0410) are 100% similar to either "a"(U+0061) or "A"(U+0041) in UC-SimList_v. (We consider each character is semantically similar to itself and calculate the similarity of a given pair of characters by multiplying visual similarity and semantic similarity).

About KDE

//Wan: please
Talk about the basic theory of KDE. How is the performance in the former usage and investigation? Cite the paper you submitted?
What's the interface to apply the method if you want to use this method? (e.g. the two arrays under evaluation should have the same dimension)
Provide the source code to the web (http://antiphishing.cs.cityu.edu.hk) tell people to download if want to use.
Give an example of calculating the similarity of "a" (U+0061) and "а" (U+0430)
//

4. KDE based UC-SimList Generation

The UC-SimList generation includes the UC-SimList_s construction, UC-SimList_v construction, and a process of generation UC-SimList using the two constructed lists.
The construction of UC-SimList_s needs a complete survey on all languages used UCS. In many cases, we can find corresponding replacement to one character, such as "a" to "A" (English in lower-case and upper case), "银" to "銀" (Chinese in simplified-form and traditional-form), "あ" to "ア" (Japanese in hirakana and katakana). The investigation on constructing UC-SimList_s is heavily depending the language usage and character representation on the semantic level of each language character set in UCS, such as "一" and "壹" (stands for "one" in Chinese), and there is no automatic way to construct it. So UC-SimList_s has to be constructed manually. We constructed the basic version of UC-SimList_s which includes English, Chinese, and Japanese.
The construction of UC-SimList_v needs the character similarity assessment metrics, where we choose KDE. KDE is an excellent for character similarity evaluation as discussed Section 0. Arial Unicode MS font 1.01 [9][10] is the most complete font we can find, and it covers the largest number of characters among all available fonts in the world. So we choose it to be the font in our experiments and the UC-SimList_v construction. Arial Unicode MS 1.01 is a true type font (TTF). Each character is represented with

many/a contour(s); each contour comprises quadratic spline(s) (QS) and/or straight line(s) (SL); and each QS/SL is represented with critical points. We retrieve the font information (the critical points of each character) from Arial Unicode MS 1.01 and convert them with sets of points.

We denote N to be the number of points in the converted point sets. The N the larger, the quality of the representation will be the better, as shown in Figure 5. Experiment shows that, when N=50, it is good enough to represent the visible characters in the range of U+0000 to U+00FF (ie. ASCII). Experiment also shows that the process of calculating the KDEs for one character to the other ones (2^16-1=65535 characters) takes about 1 hour when N=100, such that we need $\frac{1}{2^{16}-1}C_{2^{16}}^{2}$ hours (about 3.74 years) to finish the calculation (We used a P4 2.4G PC with 1G memory). So it is unrealistic to calculate a complete UC-SimList_v. As a matter of fact, it will take much longer to calculate if we concern about the information lose and use N=200. Experiments shows that N=200 will be good enough to represent all characters in the UCS. Figure 5 shows one example of representing Chinese character “银” using 200 points. So we only calculate the characters in the range of U+0000 to U+00FF (ASCII). In fact these characters are the most frequently used characters and most easily to be targeted to carry out *Unicode attacks*.

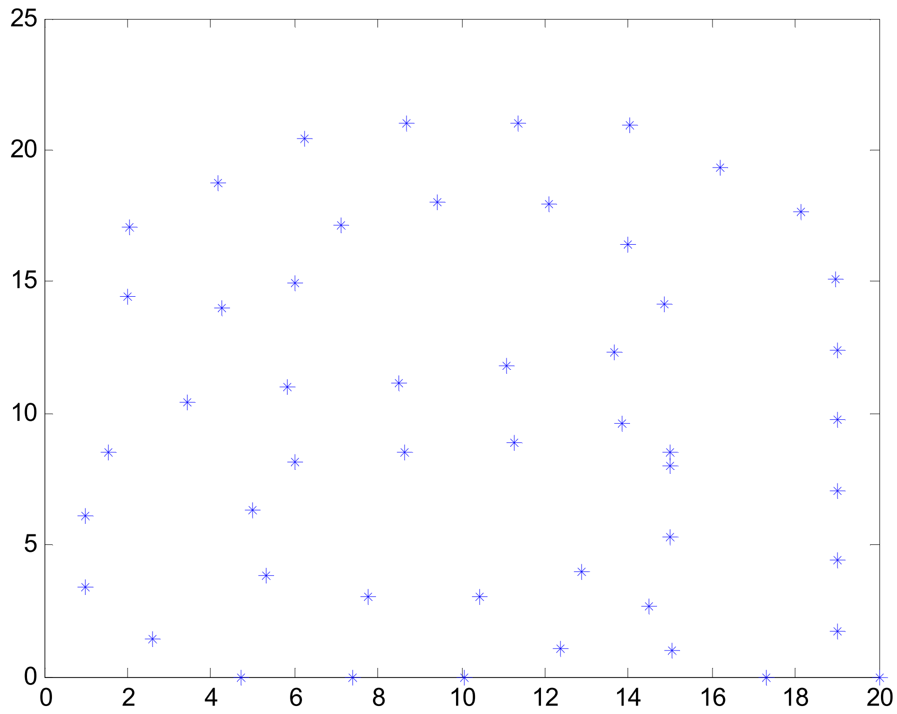
The utilization of KDE brings us the accuracy improvement comparing with the pixel-overlapping based assessment [14] [15], where U+94F6:银 and U+9512:锒 are considered to be similar. However, U+9512:锒 ranked the eleventh using that method as shown in Figure 5 because the two characters’ glyphs have some offset to each other, such that the common area of the two characters are reduced. In comparison with the former method, the KDE based assessment performs much better and rank U+9512:锒 to be the second similar character to U+94F6:银.

The construction of UC-SimList uses UC-SimList_s and UC-SimList_v as presented in Section 0. UC-SimList_s, UC-SimList_v, and UC-SimList are available at [2] for free for researchers.

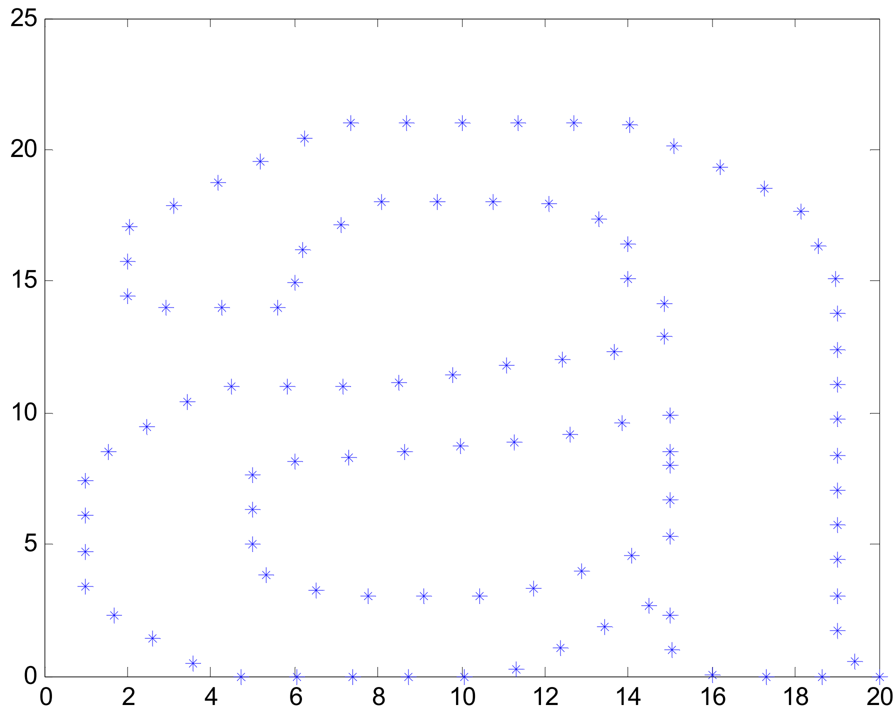
Similarity Rank	Pixel-overlapping based assessment		KDE based assessment	
	Similarity	Char Code and Char	Similarity 1-KDE	Char Code and Char
1	0.765574	U+953F:锿	0.94583	U+9503:锃
2	0.75	U+9530:锰	0.9448	U+9512:锒
3	0.746032	U+9502:锂	0.93425	U+94BD:钽
4	0.741408	U+94FB:铻	0.93394	U+953F:锿
5	0.740973	U+9491:钑	0.92788	U+94A1:钡
6	0.733114	U+9510:锐	0.92713	U+9502:锂
7	0.729299	U+88C9:裉	0.9271	U+953D:锽
8	0.728171	U+94C0:铀	0.92619	U+94BF:钿
9	0.727422	U+94C7:铇	0.91884	U+953C:锼
10	0.726524	U+9526:锦	0.91836	U+9522:锢
11	0.724194	U+9512:锒	0.91583	U+94B7:钷
12	0.72293	U+9525:锥	0.91101	U+6068:恨

Figure 4. The comparison of using pixel-overlapping based assessment and KDE based assess-

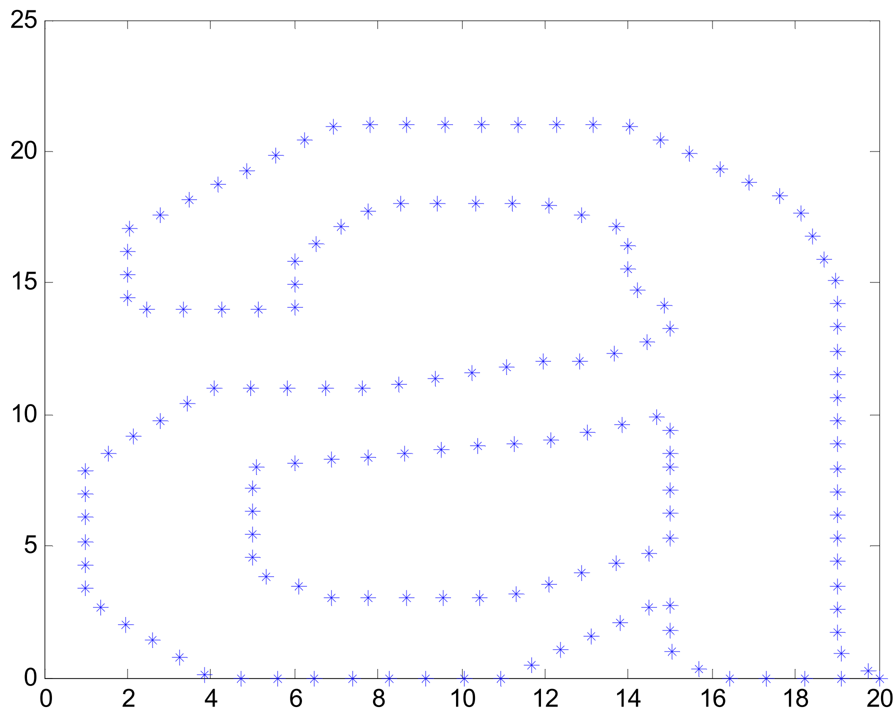
ment using sample target character, U+94F6:银 (The four digits between “U+” and “:” are the character numbers in hexadecimal form and the character is following “:”).



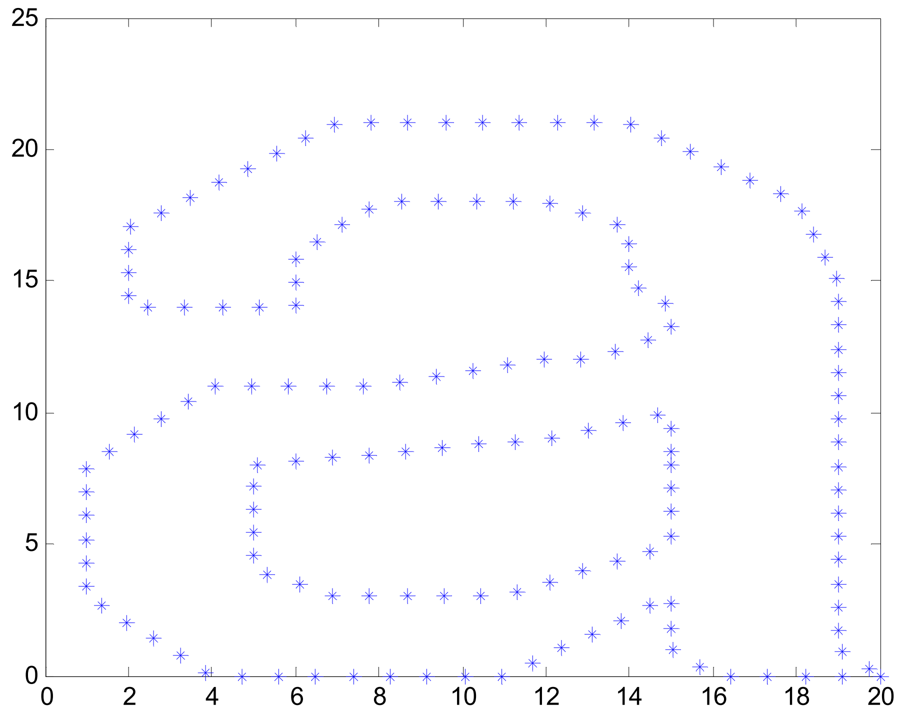
N=50



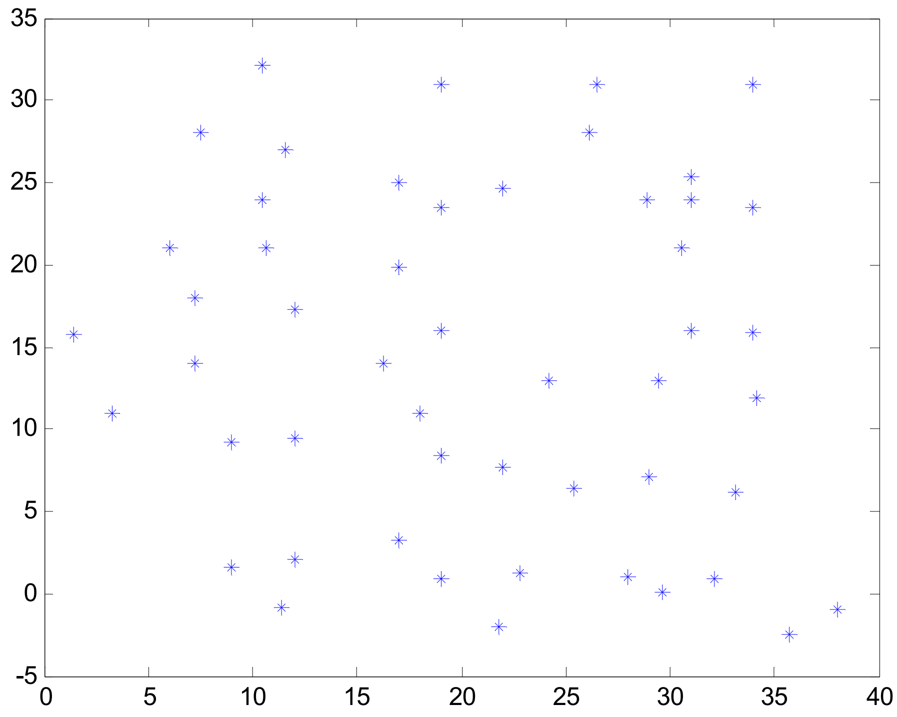
N=100



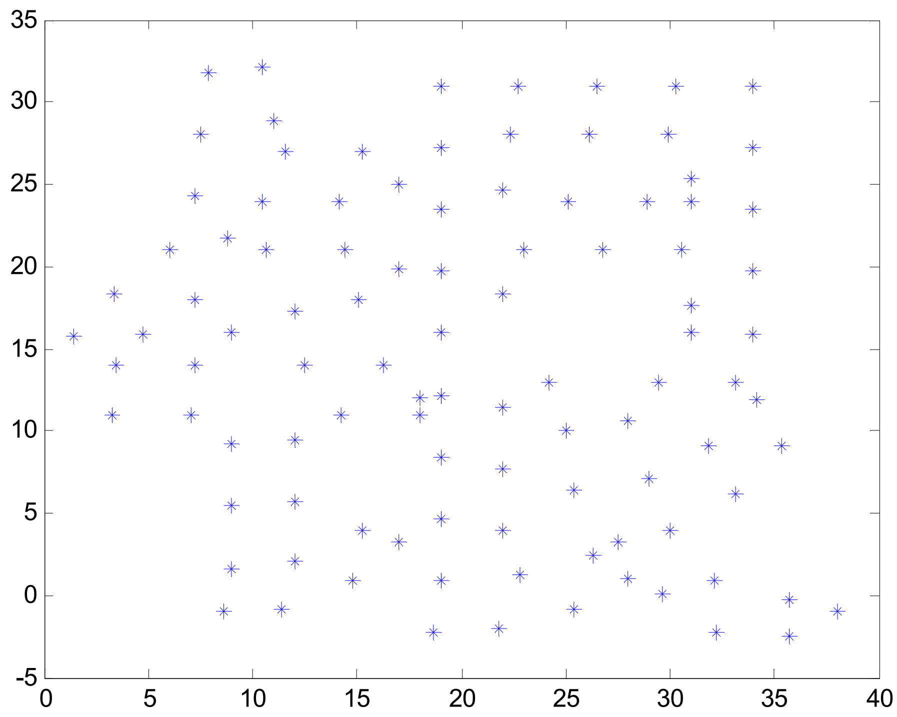
N=150



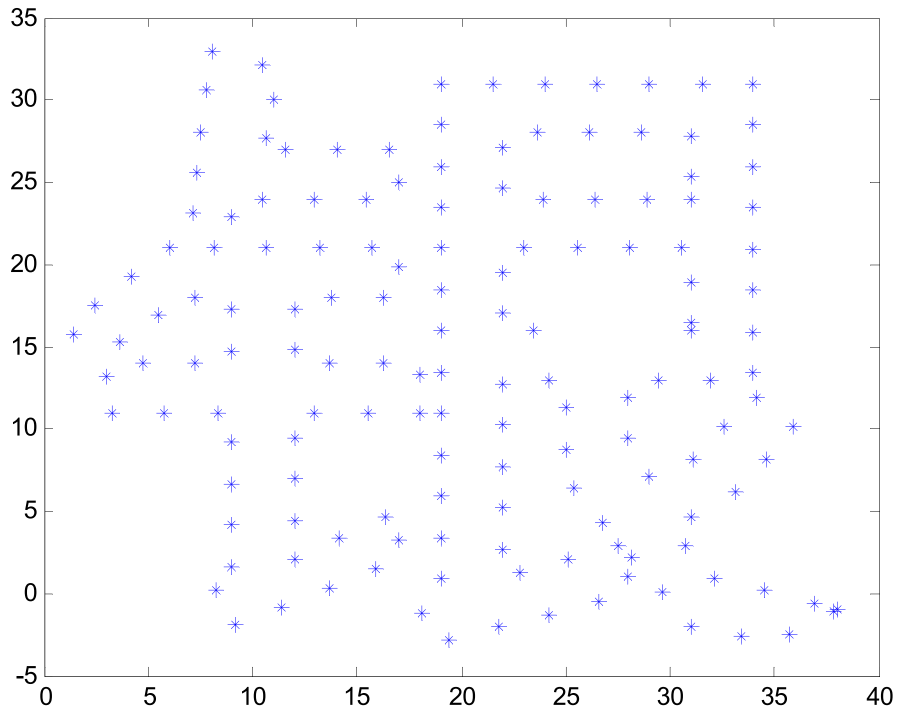
N=200



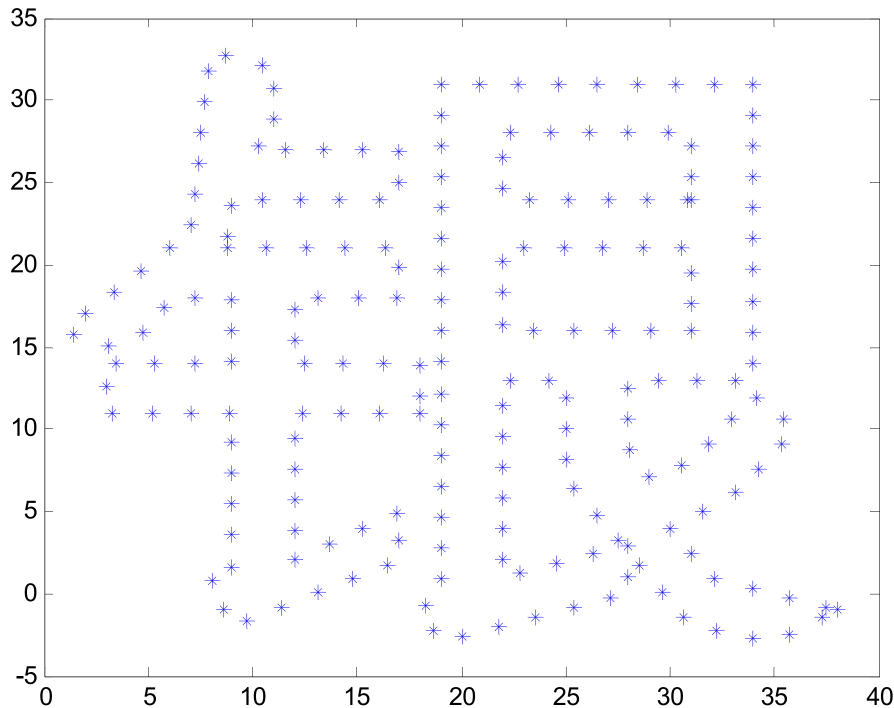
N=50



N=100



N=150



N=200

Figure 5. The converted point sets of “a” and “银” with N=50, 100, 150, and 200. We can see when N=100 it is good enough to represent “a” while when N=200, the representation of “银” could be satisfied. (Unit of X-axis and Y-axis: Font-Point)

5. Conclusion and Future Works

In this paper, we discuss the *Unicode attacks*, which could be generally classified into three categories. These attacks are essentially based on the coexistence of visual similar characters in UCS. In addition, semantically similar characters in UCS are also needed to be considered seriously. We need to assess the similarity of Unicode strings to evaluate the genuineness of a given one, so one of most basic cornerstones to detect/discover *Unicode attack* is to construct the UC-SimList. We constructed the prototype UC-SimList_s of English, Chinese, and Japanese. We also proposed an effective symbol similarity assessment measure, KDE, to constructed UC-SimList_v. Finally, we put all of the lists available for researchers on the Web [2].

The UC-SimList_s needs to be filled in and polished. That is because (1) there are character sets of many more languages in UCS other than English, Chinese, and Japanese, (2) more semantic similarity relationships need to be considered as well, for instance, we

should consider “一” and “壹” semantically similar. The UC-SimList_v construction can be done in an automatic way, however, the algorithm proposed in this paper is quite time consuming. Although we have calculated the visible characters in the ASCII, which are most frequently used, the UC-SimList_v still needs to be consummated with further effort. We can redesign the algorithm to reduce the calculation time.

6. References

[1]. ANSI, *American Standard Code for Information Interchange*, www.ansi.org.

[2]. Anti-Phishing Group of City University of Hong Kong, http://antiphishing.cs.cityu.edu.hk

[3]. Dhamija R., Tygar J. D., *The Battle against Phishing: Dynamic Security Skins*, Symposium on Usable Privacy and Security 2005, pages 77-99, 2005.

[4]. Duerst M.,Suignard M., *RFC 3987: Internationalized Resource Identifiers (IRIs)*, The Internet Society (2005), Jan. 2005.

[5]. Fu A. Y., Liu W., Deng X., *EMD based Visual Similarity for Detection of Phishing Webpages*, in Proceedings of International Workshop on Web Document Analysis 2005, 2005.

[6]. Jakobsson M., *Modeling and Preventing Phishing Attacks*, Phishing Panel of Financial Cryptography, 2005.

[7]. Liu W., Huang G., Liu X., Zhang M., Deng X., *Detection of Phishing Webpages based on Visual Similarity*, in Proceedings of the 14th International World Wide Web Conference, pages 1060-1061, 2005.

[8]. Liu W., Huang G., Liu X., Zhang M., Deng X., *Phishing Webpage Detection*, in Proceedings of 8th International Conference on Documents Analysis and Recognition, pages 560-564, 2005.

[9]. Microsoft, *Arial Unicode MS, Version 1.01*, 2000.

[10]. Microsoft, *Description of the Arial Unicode MS font in Word 2002*, http://support.microsoft.com/kb/q287247.

[11]. The Unicode Consortium, http://www.unicode.org.

[12]. www.sitewatcher.biz

[13]. Gabrilovich E. and Gontmakher A., *The Homograph Attack*, Communications of the ACM, Volume 45(2), page128, February 2002

[14]. Fu A. Y., Deng X., Liu W., *A Potential IRI based Phishing Strategy*, in the Proceedings of 6th International Conference on Web Information Systems Engineering, Lecture Notes in Computer Science Volume 3806, pages 618 - 619 , WISE 2005, New York, USA, Nov. 20-22, 2005

[15]. Fu A. Y., Deng X., Liu W., *A Methodology for Unicode String Similarity Assessment*, Technical Report, Detp. of Computer Science, City Univ. of Hong Kong, Oct. 2005

[16]. Liu Wenyin, Xiaotie Deng, Guanglin Huang, Anthony Y. Fu, *An Anti-Phishing*

Strategy based on Visual Similarity Assessment, to appear in IEEE Internet Computing, Mar/Apr. 2006

//The possibility of coexistence of the two sampled points.

$$\int_{-\infty}^{\infty}\int_{\frac{d}{2}}^{\infty}\frac{1}{(2\pi\sigma^2)\, e^{\frac{x^2+y^2}{2\sigma^2}}}\,dx\,dy$$

$$\frac{\text{Erfc}\left[\frac{d}{2\sqrt{2}\sqrt{\sigma^2}}\right]\ \text{If}\left[\text{Re}[\sigma^2]>0,\ \sqrt{2\pi}\ \sqrt{\sigma^2},\ \text{Integrate}\left[e^{-\frac{y^2}{2\sigma^2}},\ \{y,-\infty,\infty\},\ \text{Assumptions}\to\text{Re}[\sigma^2]\le 0\right]\right]}{2\sqrt{2\pi}\ \sqrt{\sigma^2}}$$

$$\frac{1}{2}\,\text{Erfc}\left[\frac{d}{2\sqrt{2}\sqrt{\sigma^2}}\right]$$

//Parameter functions of the two splines (x1(),y1()) for line1, and (x2(),y2()) for line2

$\mathbf{x1 = (xa1 - 2\,xb1 + xc1)\ t1^2 + (2\,xb1 - 2\,xa1)\ t1 + xa1}$

$xa1 + t1\,(-2\,xa1 + 2\,xb1) + t1^2\,(xa1 - 2\,xb1 + xc1)$

$\mathbf{y1 = (ya1 - 2\,yb1 + yc1)\ t1^2 + (2\,yb1 - 2\,ya1)\ t1 + ya1}$

$ya1 + t1\,(-2\,ya1 + 2\,yb1) + t1^2\,(ya1 - 2\,yb1 + yc1)$

$\mathbf{x2 = (xa2 - 2\,xb2 + xc2)\ t2^2 + (2\,xb2 - 2\,xa2)\ t2 + xa2}$

$xa2 + t2\,(-2\,xa2 + 2\,xb2) + t2^2\,(xa2 - 2\,xb2 + xc2)$

$\mathbf{y2 = (ya2 - 2\,yb2 + yc2)\ t2^2 + (2\,yb2 - 2\,ya2)\ t2 + ya2}$

$ya2 + t2\,(-2\,ya2 + 2\,yb2) + t2^2\,(ya2 - 2\,yb2 + yc2)$

//Suppose the normal error square root to be 1

σ=1

1

//Here is the function to compute the similarity of the two splines. It turns out the computation is extremely complicated, so we need approximation

$$\int_0^1\int_0^1 \frac{1}{2}\,\text{Erfc}\left[\frac{1}{2\sqrt{2}\sqrt{\sigma^2}}\ \left(\sqrt{\left((x1-x2)^2+(y1-y2)^2\right)}\right)\right]\ ((xa1-2\,xb1+xc1)\ 2\,t1+(2\,xb1-2\,xa1)+(ya1-2\,yb1+y$$

$$((xa2-2\,xb2+xc2)\ 2\,t2+(2\,xb2-2\,xa2)+(ya2-2\,yb2+yc2)\ 2\,t2+(2\,yb2-2\,ya2))\ dt2\,dt1$$

//We can do taylor expansion to simplify the process

Series[Erfc[x],{x,0,6}]

$$1-\frac{2x}{\sqrt{\pi}}+\frac{2x^3}{3\sqrt{\pi}}-\frac{x^5}{5\sqrt{\pi}}+O[x]^7$$

$$\mathbf{x}=\frac{1}{2\sqrt{2}\sqrt{\sigma^2}}\left(\sqrt{\left((x1-x2)^2+(y1-y2)^2\right)}\right)$$

$$\frac{1}{2\sqrt{2}\sqrt{\sigma^2}}\left(\sqrt{\left((xa1-xa2+t1(-2\,xa1+2\,xb1)-t2(-2\,xa2+2\,xb2)+t1^2(xa1-2\,xb1+xc1)-t2^2(xa2-2\,xb2+xc2\right.}\right.$$
$$\left.\left.(ya1-ya2+t1(-2\,ya1+2\,yb1)-t2(-2\,ya2+2\,yb2)+t1^2(ya1-2\,yb1+yc1)-t2^2(ya2-2\,yb2+yc2))^2\right)\right)$$

$$\int_0^1\int_0^1\frac{1}{2}\left(1-\frac{2x}{\sqrt{\pi}}+\frac{2x^3}{3\sqrt{\pi}}-\frac{x^5}{5\sqrt{\pi}}\right)((xa1-2\,xb1+xc1)\,2\,t1+(2\,xb1-2\,xa1)+(ya1-2\,yb1+yc1)\,2\,t1+(2\,yb1-$$
$$((xa2-2\,xb2+xc2)\,2\,t2+(2\,xb2-2\,xa2)+(ya2-2\,yb2+yc2)\,2\,t2+(2\,yb2-2\,ya2))\,dt2\,dt1$$

$$\text{expanded}=\frac{1}{2}\left(1-\frac{2x}{\sqrt{\pi}}+\frac{2x^3}{3\sqrt{\pi}}-\frac{x^5}{5\sqrt{\pi}}\right)((xa1-2\,xb1+xc1)\,2\,t1+(2\,xb1-2\,xa1)+(ya1-2\,yb1+yc1)\,2\,t1+(2$$
$$((xa2-2\,xb2+xc2)\,2\,t2+(2\,xb2-2\,xa2)+(ya2-2\,yb2+yc2)\,2\,t2+(2\,yb2-2\,ya2))$$

$$\frac{1}{2}(-2\,xa1+2\,xb1+2\,t1(xa1-2\,xb1+xc1)-2\,ya1+2\,yb1+2\,t1(ya1-2\,yb1+yc1))$$
$$(-2\,xa2+2\,xb2+2\,t2(xa2-2\,xb2+xc2)-2\,ya2+2\,yb2+2\,t2(ya2-2\,yb2+yc2))$$
$$\left(1-\frac{1}{640\sqrt{2\pi}\,(\sigma^2)^{5/2}}\left(\left((xa1-xa2+t1(-2\,xa1+2\,xb1)-t2(-2\,xa2+2\,xb2)+t1^2(xa1-2\,xb1+xc1)-t2^2(xa2-\right.\right.\right.$$
$$\left.(ya1-ya2+t1(-2\,ya1+2\,yb1)-t2(-2\,ya2+2\,yb2)+t1^2(ya1-2\,yb1+yc1)-t2^2(ya2-2\,yb2+yc2))^2\right)^{5/2}$$
$$\frac{1}{24\sqrt{2\pi}\,(\sigma^2)^{3/2}}\left(\left((xa1-xa2+t1(-2\,xa1+2\,xb1)-t2(-2\,xa2+2\,xb2)+t1^2(xa1-2\,xb1+xc1)-t2^2(xa2-2\,xb\right.\right.$$
$$\left.(ya1-ya2+t1(-2\,ya1+2\,yb1)-t2(-2\,ya2+2\,yb2)+t1^2(ya1-2\,yb1+yc1)-t2^2(ya2-2\,yb2+yc2))^2\right)^{3/2}$$
$$\frac{1}{\sqrt{2\pi}\sqrt{\sigma^2}}\left(\sqrt{\left((xa1-xa2+t1(-2\,xa1+2\,xb1)-t2(-2\,xa2+2\,xb2)+t1^2(xa1-2\,xb1+xc1)-t2^2(xa2-2\,xb2+\right.}\right.$$
$$\left.\left.\left.(ya1-ya2+t1(-2\,ya1+2\,yb1)-t2(-2\,ya2+2\,yb2)+t1^2(ya1-2\,yb1+yc1)-t2^2(ya2-2\,yb2+yc2))^2\right)\right)\right)$$

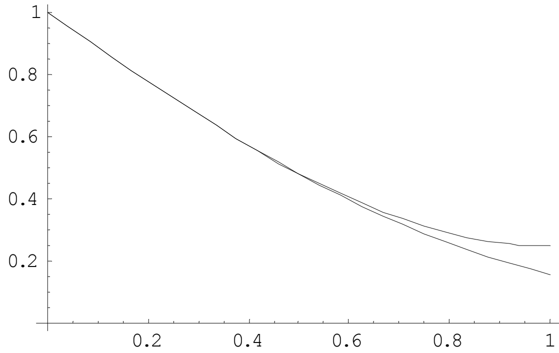
```
Series[expanded,{t1,0,2},{t2,0,2}]
```

```
s=Series[Erfc[x],{x,0,3}]
```

$$1-\frac{2x}{\sqrt{\pi}}+\frac{2x^3}{3\sqrt{\pi}}+O[x]^4$$

// Should adjust d to the range of $[0,2\sqrt{2}\,\sigma]$, such that $\frac{d}{2\sqrt{2}\sqrt{\sigma^2}}$ could be in the range of [0, 1], as shown in the follows

```
tt={Normal[s],Erfc[x]};
pp=Plot[Evaluate[tt],{x,0,1}]
```



-Graphics-

$$\text{expanded} = \frac{1}{2}\left(1 - \frac{2x}{\sqrt{\pi}} + \frac{2x^3}{3\sqrt{\pi}}\right)((xa1 - 2\,xb1 + xc1)\,2\,t1 + (2\,xb1 - 2\,xa1) + (ya1 - 2\,yb1 + yc1)\,2\,t1 + (2\,yb1 - 2\,ya$$

$$((xa2 - 2\,xb2 + xc2)\,2\,t2 + (2\,xb2 - 2\,xa2) + (ya2 - 2\,yb2 + yc2)\,2\,t2 + (2\,yb2 - 2\,ya2))$$

$$\frac{1}{2}(-2\,xa1 + 2\,xb1 + 2\,t1\,(xa1 - 2\,xb1 + xc1) - 2\,ya1 + 2\,yb1 + 2\,t1\,(ya1 - 2\,yb1 + yc1))$$

$$(-2\,xa2 + 2\,xb2 + 2\,t2\,(xa2 - 2\,xb2 + xc2) - 2\,ya2 + 2\,yb2 + 2\,t2\,(ya2 - 2\,yb2 + yc2))$$

$$\Bigg(1 + \frac{1}{24\sqrt{2\pi}\,(\sigma^2)^{3/2}}\,(((xa1 - xa2 + t1\,(-2\,xa1 + 2\,xb1) - t2\,(-2\,xa2 + 2\,xb2) + t1^2\,(xa1 - 2\,xb1 + xc1) - t2^2\,(xa2 - 2$$

$$(ya1 - ya2 + t1\,(-2\,ya1 + 2\,yb1) - t2\,(-2\,ya2 + 2\,yb2) + t1^2\,(ya1 - 2\,yb1 + yc1) - t2^2\,(ya2 - 2\,yb2 + yc2))^2)^{3/2}$$

$$\frac{1}{\sqrt{2\pi}\,\sqrt{\sigma^2}}\,(\sqrt{(}(xa1 - xa2 + t1\,(-2\,xa1 + 2\,xb1) - t2\,(-2\,xa2 + 2\,xb2) + t1^2\,(xa1 - 2\,xb1 + xc1) - t2^2\,(xa2 - 2\,xb2 +$$

$$(ya1 - ya2 + t1\,(-2\,ya1 + 2\,yb1) - t2\,(-2\,ya2 + 2\,yb2) + t1^2\,(ya1 - 2\,yb1 + yc1) - t2^2\,(ya2 - 2\,yb2 + yc2))^2))\Bigg)$$

```
Series[expanded,{t1,0,2},{t2,0,2}]

////////////////////////////////////////////////////////////
///////////////////////////
//This is the definition of a, which will be used in the
following function definition from former researchers.
```

$$a = \frac{8}{3\pi}\,\frac{\pi - 3}{4 - 3\pi}$$

$$\frac{8\,(-3 + \pi)}{3\,(4 - 3\pi)\,\pi}$$

```
a=0.14
0.14
```

$$\text{ApproxErf} = \text{Function}\left[x, \left(1 - e^{\left(-x^2\,\frac{\frac{4}{\pi} + a x^2}{1 + a x^2}\right)}\right)^{\frac{1}{2}}\right]$$

$$\text{Function}\left[x, \sqrt{1 - e^{-\frac{x^2\left(\frac{4}{\pi} + a x^2\right)}{1 + a x^2}}}\right]$$

```
ApproxErfc= Function[ x , 1 - ApproxErf[x]]
Function[x,1-ApproxErf[x]]
ApproxErfc[x]
```

$$1-\sqrt{1-e^{-\frac{x^2\left(\frac{4}{\pi}+0.14x^2\right)}{1+0.14x^2}}}$$

//Following part is an example

ApproxErfc[1]

$$1-\sqrt{1-e^{-\frac{\frac{4}{\pi}+\frac{8(-3+\pi)}{3(4-3\pi)\pi}}{1+\frac{8(-3+\pi)}{3(4-3\pi)\pi}}}}$$

N[%]

0.150409

x1 = (xa1 - 2 xb1 + xc1) t1^2 + (2 xb1 - 2 xa1) t1 + xa1

xa1 + t1 (-2 xa1 + 2 xb1) + t1^2 (xa1 - 2 xb1 + xc1)

y1 = (ya1 - 2 yb1 + yc1) t1^2 + (2 yb1 - 2 ya1) t1 + ya1

ya1 + t1 (-2 ya1 + 2 yb1) + t1^2 (ya1 - 2 yb1 + yc1)

x2 = (xa2 - 2 xb2 + xc2) t2^2 + (2 xb2 - 2 xa2) t2 + xa2

xa2 + t2 (-2 xa2 + 2 xb2) + t2^2 (xa2 - 2 xb2 + xc2)

y2 = (ya2 - 2 yb2 + yc2) t2^2 + (2 yb2 - 2 ya2) t2 + ya2

ya2 + t2 (-2 ya2 + 2 yb2) + t2^2 (ya2 - 2 yb2 + yc2)

σ=1

1

//Here is the function

$$\int_0^1\int_0^1 \frac{1}{2}\,\text{ApproxErfc}\left[\frac{1}{2\sqrt{2}\sqrt{\sigma^2}}\left(\sqrt{\left((x1-x2)^2+(y1-y2)^2\right)}\right)\right]((xa1-2\,xb1+xc1)\,2\,t1+(2\,xb1-2\,xa1)+(ya1-2$$
$$((xa2-2\,xb2+xc2)\,2\,t2+(2\,xb2-2\,xa2)+(ya2-2\,yb2+yc2)\,2\,t2+(2\,yb2-2\,ya2))\,dt2\,dt1$$

$$\text{expanded}=\frac{1}{2}\,\text{ApproxErfc}\left[\frac{1}{2\sqrt{2}\sqrt{\sigma^2}}\left(\sqrt{\left((x1-x2)^2+(y1-y2)^2\right)}\right)\right]$$
$$((xa1-2\,xb1+xc1)\,2\,t1+(2\,xb1-2\,xa1)+(ya1-2\,yb1+yc1)\,2\,t1+(2\,yb1-2\,ya1))$$
$$((xa2-2\,xb2+xc2)\,2\,t2+(2\,xb2-2\,xa2)+(ya2-2\,yb2+yc2)\,2\,t2+(2\,yb2-2\,ya2))$$

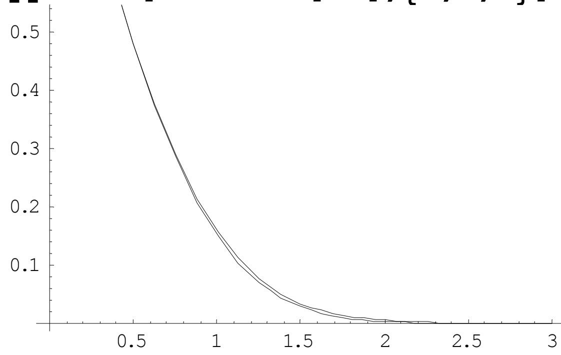
$$\frac{1}{2}\left(1-\sqrt{\left(1-e^{\frac{\left(-(xa1-xa2+t1\,(-2\,xa1+2\,xb1)-t2\,(-2\,xa2+2\,xb2)+t1^2\,(xa1-2\,xb1+xc1)-t2^2\,(xa2-2\,xb2+xc2))^2-(ya1-ya2+t1\,(-2\,ya1+2\,yb1)-t2\,(-2\,ya2+2\,yb2)+t1^2\,(\right.}{8\,(1+0.0175\,((xa1-xa2+t1\,(-2\,xa1+2\,xb1)-t2\,(-2\,xa2+}}\right)}\right)\,(-2\,xa1+2\,xb1+2\,t1\,(xa1-2\,xb1+xc1)-2\,ya1+2\,yb1+2\,t1\,(ya1-2\,yb1+yc1))$$
$$(-2\,xa2+2\,xb2+2\,t2\,(xa2-2\,xb2+xc2)-2\,ya2+2\,yb2+2\,t2\,(ya2-2\,yb2+yc2))$$

$$\int_0^1\int_0^1 \text{expanded}\,dt2\,dt1$$

```
Series[expanded,{t1,0,2},{t2,0,2}]

//Here is the demo to show the approximation of Erfc[] works
 tt={Erfc[x],ApproxErfc[x]};

pp=Plot[Evaluate[tt],{x,0,3}]
```



-Graphics-