Soundex-algoritmer
Hur bra fungerar de på svenska namn?

Patrik Hansson

Examensarbete för 15 hp
Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds universitet
Thesis for a diploma in computer science, 15 ECTS credits
Department of computer science, Faculty of science, Lund University

Soundex-algoritmer
Hur bra fungerar de på svenska namn?

Sammanfattning

Detta arbete syftade till att undersöka befintliga soundex-algoritmer för att se hur bra de fungerar på svenska förnamn. De befintliga soundex-algoritmer som finns är huvudsakligen utvecklade för det engelska språket.
För att utföra tester på de olika algoritmerna utvecklades ett program som mot testdata sökte efter specifika namn och beräknade precision och täckning med hjälp av referensdata för de olika algoritmerna.
Resultatet av undersökningen visar att algoritmerna har en bra täckning men inte så bra precision. Förenklad Soundex har den bästa täckning och NYSIIS har den bästa precisionen mot mina testdata. Den algoritm som fick bäst F-mått var NYSSIS. Referensalgoritmen Levenshtein distance fick också bra resultat men den lider av sin dåliga prestanda och kan inte användas mot en större datamängd.
Algoritmerna fungerade bättre än förväntat, men vid fortsatt arbete borde vi kunna förbättra precisionen för svenska namn genom att göra anpassningar på någon av algoritmerna till det svenska språket.

Soundex algorithms
How well do they work on Swedish names?

Abstract

This work aimed to investigate existing Soundex algorithms to see how well they work on Swedish first names. The existing Soundex algorithms are all mainly developed for the English language.
To perform the tests on the different algorithms, we developed a program to carry out a corpus search for specific names and we evaluated their precision and recall with the help of a reference corpus for the different algorithms.
The result of the investigation shows that the algorithms have a good recall but a poor precision. Simplified Soundex has the best recall and NYSIIS has the best precision against our corpus. The algorithm that obtained the best F-measure was NYSIIS. The reference algorithm Levenshtein distance did also show good results but it suffers from its bad performance and can not be used with a large corpus.
The algorithms work better than expected, but with further work we should be able to improve the precision for Swedish names by adapting one of the algorithms to the Swedish language.

Innehållsförteckning

Förord

Jag vill tacka min handledare Pierre Nugues vid institutionen för datavetenskap, Lunds Universitet för all hjälp.
Jag vill också tacka Niklas Hertzman på företaget HH Digiarkiv AB i Malmö för att ha ställt en del av deras material till förfogande.
Tack också till alla andra som hjälpt till med detta examensarbete.

1 Bakgrund

1.1 Introduktion

Soundex-algoritmer som det kallas av de flesta heter egentligen fonetiska algoritmer. En fonetisk algoritm används för att indexera namn eller ord som de uttalas. Målet för algoritmen är att koda namn som uttalas på samma sätt med samma kod för att kunna få träff på namn trots små skillnader i stavningen. Detta sker genom att alla namn är kodade i databasen med sin soundex-nyckel och vid sökning kodas namnet som man söker efter med sin soundex-nyckel. Resultatet från sökningen blir då alla namn som har samma soundex-nyckel. De flesta algoritmer är baserade på det engelska språket.
Jag vill i det här arbetet undersöka hur bra de befintliga algoritmer som finns fungerar på det svenska språket då speciellt svenska förnamn. Ett vanligt användningsområde för den här typen av algoritmer är sökning efter namn i databaser och jag har valt att utföra undersökning i en genealogisk databas med namn som är stavade bokstavstroget från kyrkböcker från ca år 1700 och till nutid. Jag vill då se hur bra de algoritmer som finns klarar av att hitta de äldre stavningsvarianter som finns när man söker efter namn enligt dagens stavning.

1.2 Tidigare arbeten

Det finns inte många jämför bara tester men jag har hittat en utförd av Randall Wilson (2005) som utfört tester på en genealogisk databas med 178 880 personer med 25 000 identifierade matchande par där man kontrollerat täckningen för sökning både med för förnamn, efternamn och kombinationen av för- och efternamn för ett antal metoder.

1.3 Algoritmer

Här kommer jag att beskriva de olika algoritmer som ska undersökas. Jag har valt namnet Christer som exempelnamn för att visa att kodningen blir olika med de olika algoritmerna. Flera av algoritmerna har små skillnader så de ger samma kodning för enklare namn.

Ursprunglig Soundex

Ursprunglig Soundex var den första algoritmen av den här typen och utvecklades av Robert C Russell och Margaret Odell och patenterades 1918 (Russel) och 1922 (Russel). Den utvecklades för att indexera amerikanska folkräkningar. Tabell 1 visar de ursprungliga klasserna till algoritmen.

1	*a, e, i, o, u, y*	5	*l*
2	*b, f, p, v*	6	*m*
3	*c, g, k, q, s, z*	7	*n*
4	*d, t*	8	*r*

Tabell 1 Klasser för den ursprungliga algoritmen

Algoritmen såg ut så här:

1. Behåll första bokstaven
2. Slå ihop alla på varandra följande bokstäver av samma klass

3. Ta bort alla *gh, h* och avslutande *s* eller *z* och alla vokaler utom den första

Det finns ingen specificerad längd på nyckeln. Exempel på nyckel från patentansökningen är att *Highfield* ger **H1254,** *Christer* ger **C81348.** Den ursprungliga algoritmen är inte med i undersökningen.

Förenklad Soundex

Denna algoritm finns beskriven i The Art of Computer Science First Edition (Knuth 1973, pp. 391-2.). Tabell 2 visar klasserna för den förenklade soundex-algoritmen:

0	*a, e, i, o, u, y,w,h*	4	*l*
1	*b, f, p, v*	5	*m,n*
2	*c, s, k,g,j,q,x,z*	6	*r*
3	*d, t*		

Tabell 2 Klasser för förenklad soundex

Algoritmen ser ut så här i något omarbetad form:

1. Kom ihåg första bokstaven.
2. Konvertera varje bokstav (även den första) enligt tabell 2.
3. Ändra alla på varandra följande dubbla siffror till en, t.ex. ändra 22 till 2
4. Byt ut den första siffran mot bokstaven du kom ihåg i steg 1.
5. Radera alla nollor.
6. Justera till fyra tecken genom trunkering eller paddning till höger med nollor.

Namnet *Christer* ger koden **C623** med denna algoritm

Miracode

Den här algoritmen är identisk med förenklad Soundex förutom en extra regel. Den användes vid indexering av 1910 år amerikanska folkräkning och den underhålls av National Archives i USA (National Archives). Den kallas även Amerikansk Soundex.
Den extra regeln är att enstaka h eller w som separerar konsonanter ignoreras, så att om konsonanterna tillhör samma klass slås de samman. *Ashcraft* kodas som A-261 och inte A-226 som i förenklad Soundex. Denna algoritm är också beskriven i The Art of Computer Science Second Edition (Knuth 1998, pp. 394-5). Soundex-koden för ett namn innehåller en bokstav följd av tre siffror. Bokstaven är den första bokstaven i namnet och siffrorna kodas utifrån de kvarvarande konsonanterna. Likljudande konsonanter kodas på samma sätt t.ex. *B, F, P* och *V* kodas alla till 1. Vokaler kan påverka kodningen men kodas aldrig direkt förutom om de finns som första bokstav i namnet. Den enda ändring som är gjord av mig är att jag lagt till *å, ä, ö* som vokaler. Den exakta algoritmen är:

1. Behåll första bokstaven
2. Ta bort alla av de följande bokstäverna, förutom om de är den första bokstaven: *a, e, h, i, o, u, å, ä, ö, w, y*
3. Tilldela de kvarvarande bokstäverna siffror enligt tabell 2:

1	*b, f, p, v*	4	*L*
2	*c, g, j, k, q, s, x, z*	5	*m, n*
3	*d, t*	6	*r*

Tabell 3 Klasser för Miracode

4. Om två eller flera bokstäver med samma nummer står intill varandra i det ursprungliga namnet, uteslöt då alla utom den första.
5. Returnera de första fyra tecknen. Padda från höger med nollor om det är färre tecken än fyra.

Namnet *Christer* ger koden **C623** med denna algoritm

SQL Server Soundex

Denna algoritm skiljer sig från den ursprungliga genom att endast ignorerar intilliggande dubbla fonetiska ljud och dessutom ignorerar den inte bokstaven om den är dubbel tillsammans med den första bokstaven. T.ex. SQL Server kodar *PPPP* som **P100** medan miracode kodar den till **P000**. Namnet *Christer* får koden **C623**. Denna variant är den som finns implementerad i Microsofts SQL Server.

Dubbel Metaphone

Dubbel Metaphone (Philips 2000) är en vidareutveckling av Metaphone (Philips 1990) skapad av Lawrence Philips. Algoritmen publicerade i juni år 2000 och har en mycket mer komplicerad regeluppsättning. Den kan även returnera en alternativ kodning för ord att som kan uttalas på olika sätt för att ta hänsyn till uttal för andra språk. Den enda ändring som är gjord av mig är att jag lagt till *å, ä, ö* som vokaler. *Christer* ger för denna algoritm koden **KRST.**

Daitch-Mokotoff

Daitch-Mokotoff (Mokotoff 1985; Daitch 1986) skapades av Gary Mokotoff och förbättrades av Randy Daitch båda från judiska genealogiska föreningen. Denna algoritm är en utveckling från Soundex för att hantera germanska och slaviska namn. Förbättringar som gjorts mot Soundex är:

- Kodade namn är sex siffror lång, resulterande i större precision
- Kodade namn kan sparas som numeriska värde, vilket kan spara plats i vissa applikationer.
- Flera regler kan koda multipla bokstavskombinationer till en siffra.
- Multipla möjliga kodningar returneras för ett namn.

Reglerna för Daitch-Mokotoff följer nedan:

1. Namn kodas till sex siffror, varje siffra representerar ett ljud som listas i kodningstabellen i bilaga 1.
2. Bokstäverna *A, E, I, O, U, J* och *Y* kodas alltid i början på ett namn, som i *A*ugsburg (*0*54795). I alla andra situationer ignoreras de förutom om två av dem bildar ett par före en vokal, som i Br*eu*er (79*1*900), men inte Fr*eu*d. Bokstaven *H*

kodas I början av namn, som i *H*alberstadt (*5*87943) eller om den kommer före en vokal som i Mann*h*eim (66*5*600), annars kodas den inte.

3. När intilliggande ljud kan kombineras till ett större ljud ges de kodnumret för det större ljudet, som i Chernowitz, som inte kodas som Chernowi-*t*- *z* (496*734*) utan som Chernowi-*tz* (496*74*0).
4. När intilliggande bokstäver har samma kodnummer, kodas de som ett ljud, som i Cherkassy, som inte kodas som Cherka-*s*-*s*y (495*44*0) utan som Cherkassy (495400). Undantag för denna regel är bokstavskombinationen *MN* och *NM* vars bokstäver kodas separat, som i Klei*nm*an som kodas till 5*86*6*6*0 inte till 5*86*600.
5. När namn består av fler en ett ord, kodas det som ett ord, som Nowy Targ, vilket behandlas som Nowytarg.
6. Flera bokstäver och bokstavskombinationer kan låta på ett av två sätt. Bokstäverna och bokstavskombinationerna *CH*, *CK*, *C*, *J* och *RZ* (Se bilaga 1), tilldelas två möjliga kodnummer. Pröva båda möjligheterna.
7. När namn inte innehåller tillräckligt många kodad ljud för att fylla de sex siffrorna, kodas resterande siffrorna med 0 som i Berlin (798*60*0) som bara har fyra kodad ljud(*B*-*R*- *L*-*N*).

Den enda ändring som är gjord av mig är att jag lagt till *å, ä, ö* som vokaler. *Christer* ger koden **594390** när man använder denna algoritm.

NYSIIS

New York State Identification and Intelligence System fonetisk kod, känd som NYSIIS, är en soundex-algoritm skapad av Robert L Taft (1970) som en del av New York State Identification and Intelligence System (nu en del av New York State Division of Criminal Justice Services). Den har en ökad exakthet med 2,7 % över Miracode. Algoritmen ser ut enligt nedan:

1. Översätt de första bokstäverna i namnet: *MAC → MCC, KN → NN, K → C, PH → FF, PF → FF, SCH → SSS*
2. Översätt de sista bokstäverna i namnet: *EE → Y, IE → Y, DT, RT, RD, NT, ND → D*
3. Första bokstaven i nyckeln = första bokstaven i namnet
4. Översätt återstående bokstäver enligt följande regler, öka med en bokstav varje gång:
   - *EV → AF* eller *A, E, I, O, U, Å, Ä, Ö → A*
   - *Q → G, Z → S, M → N*
   - *KN → N* eller *K → C*
   - *SCH → SSS, PH → FF*
   - *H →* Om bokstaven före eller efter inte är en vokal, ta bort den.
   - *W →* Om bokstaven före är en vokal, ta bort den.
   - Lägg till nuvarande bokstav till nyckeln om den nuvarande nyckeln inte är samma som den senaste nyckelbokstaven.
5. Om den sista bokstaven är *S*, ta bort den.
6. Om den sista bokstaven är *AY*, ändra till *Y*.
7. Om sista bokstaven är *A*, ta bort den.

Den enda ändring som är gjord av mig är att jag lagt till *å, ä, ö* som vokaler. *Christer* ger här **CHRASTAR** som kod.

Phonix

Phonix (Gadd 1990) är ytterligare en variant av Soundex som är publicerad första gången 1988. Här använder man ungefär samma regler som ursprunglig Soundex men har dessutom ca 160 olika substitutioner som görs på ordet före värdet beräknas se bilaga 2. Algoritmen ser ut enligt nedan:

1. Utför fonetiska substitutioner:
    - Endast de specificerade bokstäverna tas bort
    - Substitutioner utförs i en bestämd ordning
    - Utför alla förekomster av substitutionen inom strängen innan du fortsätter med nästa substitution
    - Resultatet av substitutionen kan skapa nya strängar som innehåller nya substitutioner för efterföljande regler
2. Behåll den första bokstaven för kodnyckeln
3. Byt ut mot v om *A, E, I, O, U,* eller *Y*
4. När strängen slutar på med *ES* ta bort *E*
5. Lägg till ett *E* om strängen slutar *med A, I, O, U,* eller *Y*
6. Ta bort den sista bokstaven vad den än är
7. Ta bort den nya sista bokstaven om den inte *är A, E, I, O, U,* eller *Y*
8. Upprepa punkt 7 tills det vi kommer till en vokal. Detta resulterar i en sträng utan dess avlutande ljud.
9. Ta bort alla förekomster av A, *E, I, O, U, Y, H* och *W*
10. Ta bort den ena av alla dubbla konsonanter
11. Byt ut alla konsonanter mot deras numeriska värde
12. Lägg tillbaka den sparade första bokstaven enligt punkt 2 och 3

En exempel kod med denna algoritm ger för *Christer* **C6836000**.

List Soundex

I ett papper från IJCAI-07 av Rahul Bhagat och Eduard Hovy (2007) föreslås en ny metod kallad List Soundex som är en kombination av Förenklad Soundex och Levenshtein distance. Eftersom beräkningen av Levenshtein distance är $O(n^2)$ för längden av strängen kan man inte beräkna Levenshtein distance på hela databasen. Jag har inte implementerat den i denna undersökning eftersom min databas är för liten för att det ska ge något bättre resultat än att bara använda en av dem. Den fungerar som så att man först kör Soundex på hela databasen för att dela upp namnen i mindre delmängder som man sedan kör Levenshtein distance på.

Levenshtein distance

I informationsteori och datavetenskap är Levenshtein distance ett strängmått vilket är ett sätt att mäta avståndet mellan två strängar. Levenshtein distance mellan två strängar ges av det minsta antal operationer som krävs för att transformera den ena strängen till den

andra där en operation är en insättning, radering eller utbyte av en bokstav. Den är döpt efter Vladimir Levenshtein (Wikipedia, Levenshtein distance) som kom på detta avstånd 1965. Den är användbar i applikationer som behöver bestämma hur lika två strängar är, t.ex. för stavningsprogram. T.ex. är Levenshtein distance mellan "*Katt*" och "S*itta*" är 3, efter som dessa tre redigeringar ändrar denna ena strängen till den andra och det finns inga andra sätt att ändra strängen med färre steg än tre:

katt → satt (utbyte av 'k' till 's')
satt → sitt (utbyte av 'a' till 'i')
sitt → sitta (insättning av 'a' mot slutet)

Detta är ingen soundex-algoritm men är med som referensalgoritm. Sättet som använts i denna undersökning är att det gjorts en jämförelse mellan det sökta namnet och alla andra namn i databasen och i det fall som kallats *Levenshtein distance 1* har alla namn som har fått ett avstånd på 1 betraktats som en träff och inga andra och på motsvarande sätt har då *Levenshtein distance 2* har alla namn som fått ett avstånd på 2 betraktats som en träff.

2 Metod

2.1 Testdata

Som testdata (Corpus) användes förnamn från genealogisk data från flera olika källor. Den största mängden data kom från ett vigselregister och ett boupptäckningsregister som företaget DigiArkiv AB (DigiArkiv) var vänliga nog att ge mig tillgång till. Sedan använde jag även alla förnamn från min egen släktforskning och till sist så använde jag ett register på alla förnamn som människor döps till i Sverige mellan 1998-2006 (SCB). Datan har genomgått viss rensning av felaktiga tecken och även tagit bort dubbelnamn för att förenkla sökning.

2.2 Referensdata

För att kunna göra automatiska kontroller av data så har referensdata skapats. Referensdatan har hämtats från Demografisk Databas Södra Sverige (DDSS). De har manuellt normaliserat alla namn i deras databas till nutida stavning. Deras normalisering har använts för att kontrollera sökningar efter relevanta resultat.

2.3 Utvärderings metrik

För att få ett resultat som kan jämföra de olika algoritmerna har vi använt de metoder man använder inom området för informationssökning (Wikipedia, Information Retrieval).

För att mäta prestanda för applikationen används måtten precision**,** täckning (Recall) och F-mått.

Precisionen anger hur stor andel av de hittade dokumenten som bedöms vara relevanta för sökfrågan.

$$\Pr ecision = \frac{\{relevantadokument\} \cap \{hämtadedokument\}}{\{hämtadedocument\}}$$

Täckningen anger hur stor andel av alla relevanta dokument i samlingen som hittats.

$$Täckning = \frac{\{relevantadokument\} \cap \{hämtadedokument\}}{\{relevantadokument\}}$$

Generellt så minskar precisionen när täckningen ökar.

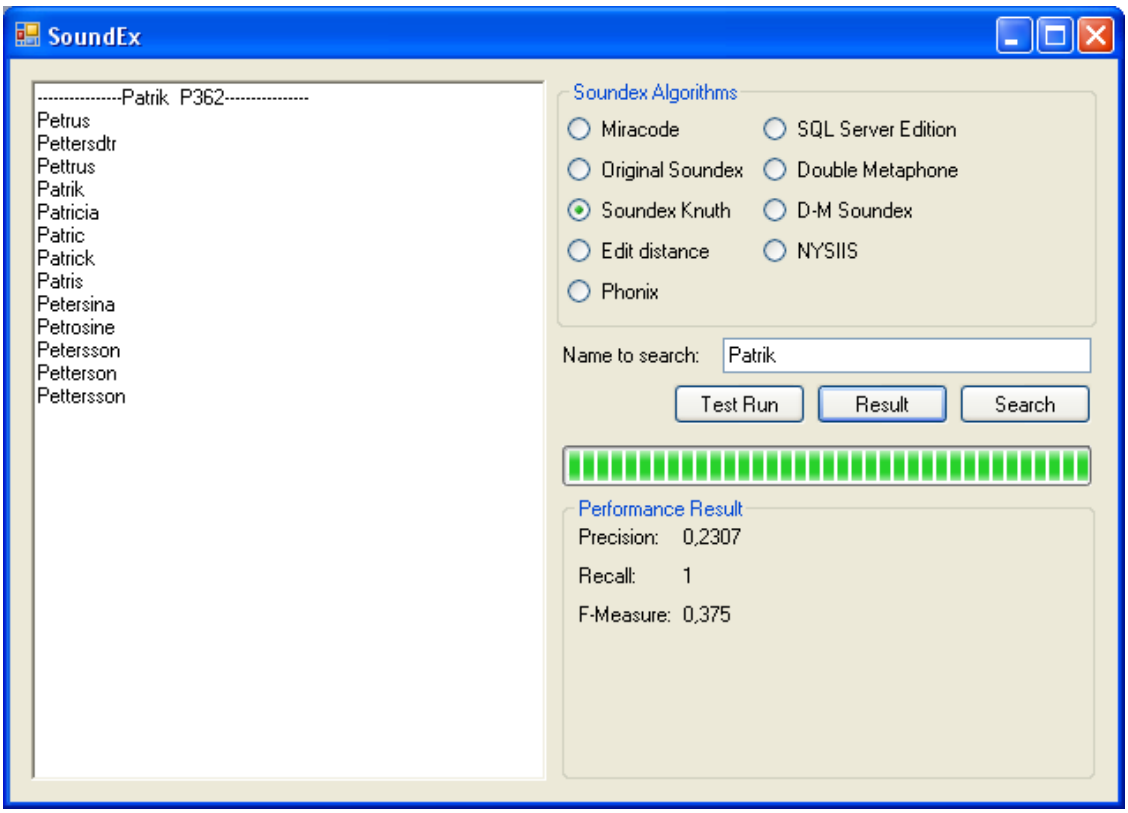
F-mått är det viktade harmoniska medelvärdet mellan Precision och täckning

$$F = 2 \times \frac{(\Pr ecision \times Täckning)}{(\Pr ecision + Täckning)}$$

2.4 Utförande

Applikation

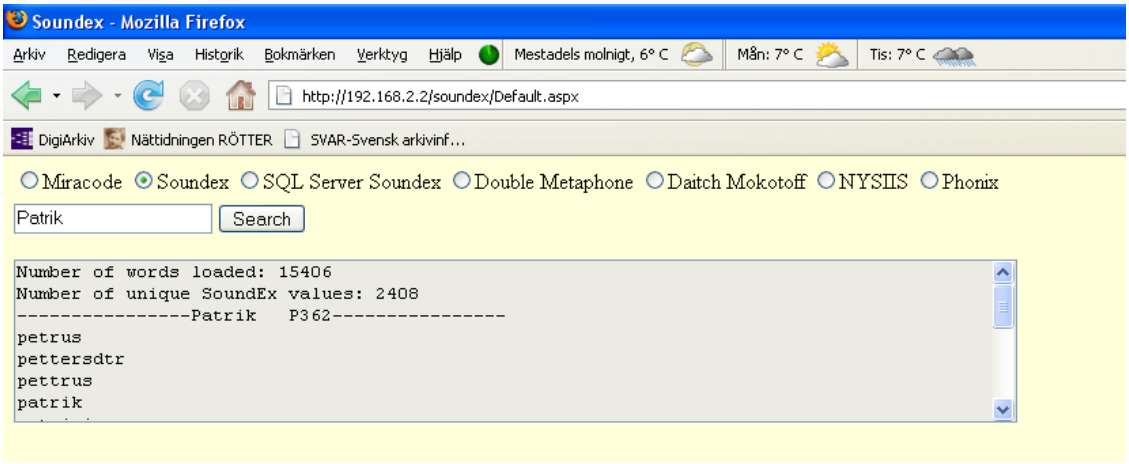
För att kunna jämföra de olika algoritmer som diskuterats tidigare så har en applikation skapats som gör en sökning efter ett antal namn och räknar ut resultat automatiskt. Programmet är skrivit i C# och .NET 3.5 i utvecklingsmiljön Microsoft Visual C# 2008 Express Edition (VC# 2008). Detta valdes för att jag är van vid denna miljö och ville testa denna nya version.
Programmet när det körs gör en sökning för varje namn i referensdatan sedan läggs antalet hämtade dokument ihop för varje sökning. På samma sätt så räknar man ut alla relevanta dokument av de hämtade dokumenten för varje sökning. Sedan räknas den totala precisionen och täckning ut för alla sökningar för att få ett mått på hur bra algoritmen är för svenska namn. Här bifogas en skärmdump av programmet.



Figur 1 Testprogrammet

Web Service

Jag har även utvecklat en enkel .NET webbservice också gjord i utvecklingsmiljön Microsoft Visual C# 2008 Express Edition som utnyttjar funktionerna som utvecklades ovan så att man kan använda dem från en webbsida. En enkel webbsida skapades i ASP som använder webbservicen Man kan där välja vilken algoritm man ska användas för sökning och sedan ange ett sök kriterium. Man får som resultat hur många ord som finns och hur många olika Soundex nycklar det finns för den valda algoritmen och förstås själva sökresultatet. En skärmdump av webbapplikationer följer nedan:



Figur 2 Webbapplikationen

3 Resultat

Resultat från Randall Wilsons undersökning ger för engelska förnamn en täckning för Miracode på 98.31 %, Dubbel Metaphone 98.09 % och NYSIIS 97.21% som en jämförelse mot mina resultat nedan.

3.1 Resultat för alla namn

Resultatet visar att efter att gjort en sökning efter alla namn som jag har referensdata för (5069 st.) för alla algoritmerna och bara använt den primära nyckeln för de algoritmer som returnerar alternativa nycklar så blev resultatet enligt nedan:

Algoritm	Precision	Täckning	F-mått
Förenklad soundex	0,1222	0,7743	0,2112
Miracode	0,1221	0,7731	0,2110
SQL Server Soundex	0,1234	0,7691	0,2127
Dubbel Metaphone	0,1024	0,7423	0,1800
Daitch-Mokotoff	0,0773	0,7688	0,1405
Phonix	0,1288	0,7117	0,2181
NYSIIS	0,2317	0,6651	0,3437
Levenshtein distance 2	0,0523	0,8706	0,0986
Levenshtein distance 1	0,3462	0,6877	0,4606

Tabell 4 Resultat för alla namn

Levenshtein distance är ingen fonetisk algoritm men är med som jämförelsealgoritm. Levenshtein distance 1 innebär att avståndet bara får vara ett från den ursprungliga strängen och Levenshtein distance 2 får då på samma sätt ha maximalt avstånd på 2.

3.2 Resultat för alla namn med fler än fem stavningsvarianter

Om man kör om testerna på samma sätt men inkluderar alla namn från referensdata som har mer än 5 stavningsvarianter(629 st.):

Algoritm	Precision	Täckning	F-mått
Förenklad Soundex	0,2582	0,6667	0,3722
Miracode	0,2579	0,6647	0,3716
SQL Server Soundex	0,2616	0,6573	0,3742
Dubbel Metaphone	0,2199	0,6205	0,3247
Daitch-Mokotoff	0,2172	0,6340	0,3236
Phonix	0,2683	0,5717	0,3652
NYSIIS	0,3940	0,4864	0,4354
Levenshtein distance 2	0,1476	0,7674	0,2476
Levenshtein distance 1	0,5146	0,4753	0,4942

Tabell 5 Resultat för alla namn med fler än fem stavningsvarianter

Levenshtein distance är ingen fonetisk algoritm men är med som jämförelsealgoritm. Levenshtein distance 1 innebär att avståndet bara får vara ett från den ursprungliga strängen och Levenshtein distance 2 är då avståndet naturligtvis 2.

3.3 Resultat för enstaka namn

Jag har valt ut några namn för att se hur de olika algoritmerna beter sig i en mer begränsad sökning. Namnen som jag valt är Kristoffer, Nils, Per, Karl, Kristina, Gertrud, Cecilia, Josefina

Namn	Förenklad Soundex		
	P	**R**	**F**
Kristoffer	**0,0193**	**0,1875**	**0,0350**
Nils	**0,6296**	**0,3269**	**0,4303**
Per	**0,7826**	**0,4864**	**0,6000**
Karl	**0,4000**	**0,2105**	**0,2758**
Kristina	**0,5741**	**0,5460**	**0,5597**
Cecilia	**0,7692**	**0,2362**	**0,3614**
Josefina	**0,6363**	**0,7368**	**0,6829**
Gertrud	**0,6428**	**0,6206**	**0,6315**

Namn	NYSIIS		
	P	**R**	**F**
Kristoffer	**0,8333**	**0,3125**	**0,4545**
Nils	**0,4358**	**0,3269**	**0,3736**
Per	**0,7368**	**0,3783**	**0,5000**
Karl	**0,4545**	**0,5263**	**0,4878**
Kristina	**0,4736**	**0,0552**	**0,0989**
Cecilia	**0,7826**	**0,1417**	**0,2400**
Josefina	**1**	**0,6842**	**0,8125**
Gertrud	**0,7333**	**0,1896**	**0,3013**

Namn	Miracode		
	P	**R**	**F**
Kristoffer	**0,0193**	**0,1875**	**0,0350**
Nils	**0,6296**	**0,3269**	**0,4303**
Per	**0,7826**	**0,4864**	**0,6000**
Karl	**0,4000**	**0,2105**	**0,2758**
Kristina	**0,5741**	**0,5460**	**0,5597**
Cecilia	**0,7692**	**0,2362**	**0,3614**
Josefina	**0,6800**	**0,8947**	**0,7727**
Gertrud	**0,6428**	**0,6206**	**0,6315**

Namn	Daitch-Mokotoff		
	P	**R**	**F**
Kristoffer	**0,6250**	**0,6250**	**0,6250**
Nils	**0,5806**	**0,3461**	**0,4337**
Per	**0,2982**	**0,4594**	**0,3617**
Karl	**0,1176**	**0,2105**	**0,1509**
Kristina	**0,3818**	**0,1288**	**0,1926**
Cecilia	**0,5968**	**0,6062**	**0,6015**
Josefina	**0,8095**	**0,8947**	**0,8500**
Gertrud	**0,6511**	**0,4827**	**0,5544**

Namn	Phonix		
	P	**R**	**F**
Kristoffer	**0,6666**	**0,125**	**0,2105**
Nils	**0,68**	**0,3269**	**0,4415**
Per	**0,7826**	**0,4868**	**0,6000**
Karl	**0,4000**	**0,2105**	**0,2758**
Kristina	**0,6885**	**0,2576**	**0,3750**
Cecilia	**0,8076**	**0,1653**	**0,2745**
Josefina	**1**	**0,3684**	**0,5384**
Gertrud	**0,6896**	**0,3448**	**0,4597**

Namn	Dubbel Metaphone		
	P	**R**	**F**
Kristoffer	**0,0555**	**0,8750**	**0,1044**
Nils	**0,6428**	**0,3461**	**0,4500**
Per	**0,4054**	**0,4054**	**0,4054**
Karl	**0,2162**	**0,4210**	**0,2857**
Kristina	**0,4365**	**0,6748**	**0,5301**
Cecilia	**0,6477**	**0,4488**	**0,5302**
Josefina	**1**	**0,6315**	**0,7741**
Gertrud	**0,5000**	**0,3103**	**0,3829**

Namn	Levenshtein distance 2		
	P	**R**	**F**
Kristoffer	**0,7142**	**0,3125**	**0,4347**
Nils	**0,3696**	**0,4615**	**0,3404**
Per	**0,1760**	**0,6756**	**0,2793**
Karl	**0,1162**	**0,7894**	**0,2027**
Kristina	**0,6315**	**0,1472**	**0,2388**
Cecilia	**0,5306**	**0,2047**	**0,2954**
Josefina	**0,6842**	**0,6842**	**0,6842**
Gertrud	**0,6981**	**0,6379**	**0,6666**

Resultaten ovan visar ganska tydligt att det finns skillnader mellan algoritmerna. Ett ganska tydligt resultat är att vanlig Levenshtein distance alltid fungerar bra. Problemet med den är prestanda.

4 Slutsatser

Man kan tydligt se att flera av algoritmerna ger väldigt lika resultat vilket kanske inte är så konstigt eftersom flera av algoritmerna är baserade på och är vidareutveckling av den ursprungliga soundex-algoritmen. Man kan också se att soundex-algoritmerna har bra täckning medan precisionen är ganska dålig. NYSIIS avviker lite här genom att ha en lite annorlunda fördelning av mellan precision och täckning. Vid en jämförelse mot Randall Wilsons undersökning så ger hans resultat och mina resultat samstämmigt resultat i den inbördes ordningen för täckningen mellan de jämförbara algoritmerna.
Levenshtein distance har också en annan fördelning mellan precision och täckning men det är ju en avståndsalgoritm så resultat kan ju förväntas att bli annorlunda.
List Soundex som föreslogs på IJCAI-07 ger i deras undersökning en precision på 0,68 och en täckning på 0,58 med ett F-mått på 0,58 vid de 25 bästa träffarna med Levenshtein distance. Deras resultat är inte direkt överförbart på mina data men ger en fingervisning om det kan ger ett bra resultat att kombinera algoritmerna. Tittar man på resultat från mina tabeller så kan man också se att det borde bli så eftersom Soundex ger en bra täckning och Levenshtein distance har ju om man använder avstånd på ett enligt mina tester väldigt bra precision. Jag tycker dock att man kunde ha valt en annan soundex-algoritm eftersom just Soundex inte kan skillnad på fösta bokstaven på ett namn så namn som Kristian och Cristian inte kommer att hamna i samma delmängd.

Förenklad Soundex

Förenklad soundex fungerar förvånansvärt bra för att vara så gammal (1918) och med tanke på hur enkel den är att implementera är den fortfarande ett bra val för att den har den bästa täckningen av de testade algoritmerna.

Miracode

Miracode eller amerikansk Soundex är den som används för att indexera de amerikanska folkräkningarna. Den har en bra täckning men har marginellt lägre både precision och täckning än förenklade Soundex mot min databas med svenska moderna och historiska svenska namn.

SQL Server Soundex

SQL Server Soundex som finns implementerad i en SQL server fungera också lika bra som förenklad Soundex. Den har lite sämre täckning men bättre precision vilket ger ett bättre F-mått än förenklad Soundex.

Dubbel Metaphone

Dubbel Metaphone som är vanlig i rättstavningsprogram har här både sämre precision och täckning än de övriga soundex-algoritmerna med undantag för Daitch-Mokotoff. Dubbel Metaphone kan returnera en alternativ kodning det gjordes lite tester med detta och det gjorde inga märkbara skillnader i resultat mot mina data därför har dessa resultat utelämnats.

Daitch-Mokotoff

Daitch-Mokotoff som är framtagen för att passa hebreiska och slaviska namn får här det sämsta F-måttet av alla soundex-algoritmerna den har dock en lite bättre täckning än Dubbel Metaphone. Den borde ha haft en bättre precision eftersom dess kod är sex tecken istället för fyra som de flesta andra har men det visar sig inte i mina testresultat. Det beror antagligen på att dess mer komplicerade regler inte passar för svenska namn. Daitch-Mokotoff algoritmen kan också lämna alternativ kodning detta har inte testat eftersom implementationen som använts inte stödde detta. Det kunde kanske ha förbättrat resultatet för denna algoritm.

NYSSIS

NYSIIS algoritmen har också ett resultat som skiljer sig mer från de övriga fonetiska algoritmerna genom att ha lite bättre precision vilket den ska ha enligt utvecklarna. Den har lite sämre täckning än de andra soundex-algoritmerna men bättre precision och det bästa F-måttet av soundex-algoritmerna.

Phonix

Phonix algoritmen är en av de nyare varianterna på soundex-algoritmerna. Den har den näst bästa precision efter NYSIIS av soundex-algoritmerna och ett lite bättre F-mått än de andra förutom NYSSIS.

Levenshtein distance

Avståndsalgoritmen ger att annat resultat speciellt med ett avstånd på ett vilket ger en bättre precision än soundex-algoritmerna men har då också sämre täckning. Ökar man det tillåtna avståndet för avståndsalgoritmen till två blir ju resultat genast annorlunda vilket ger en betydligt bättre täckning än soundex-algoritmerna medan precisionen rasar vilket ju är ett väntat resultat.
Nackdelen är att den är $O(n^2)$ över strängens längd. Eftersom man måste beräkna skillnaden mellan den sträng man söker och alla andra strängar i databasen vid varje sökning tar det väldigt långt tid att beräkna och kan i princip inte användas på stora databaser.

Precision

Den soundex-algoritm som ger bäst precision är NYSIIS följd av Phonix som blir tvåa knappt före de övriga sist kommer Daitch-Mokotoff.

Täckning

Den soundex-algoritm som ger bäst täckning är förenklad Soundex tätt följd Miracode den som har sämst täckning är då NYSIIS.

F-Mått

Den soundex-algoritm som har det bästa F-måttet är NYSIIS. Den näst bästa algoritmen blev Phonix och sista plats tog då Daitch-Mokotoff.

5 Framtiden

Man bör kunna göra förbättringar i algoritmerna för svenska detta arbete bör göras i samarbete med någon med lingvistiska kunskaper.
En algoritm som lämpar sig väl för att göra ändringar är Phonix där man på ett relativt enkelt sätt kan ändra i substitutionerna som utförs först genom att ändra ordningen på dem och ta bort eller lägga till nya.
Man kan också tänka sig göra fler undersökningar med fler algoritmer och andra metoder, t.ex. kombinationer av algoritmer.

Bilaga

Bilaga 1

Daitch-Mokotoff Soundex Kodningstabell

N/C = Kodas inte

Bokstäver	Alternativa bokstäver	Början på ett namn	Före en vokal	Alla andra
AI	AJ, AY	0	1	N/C
AU		0	7	N/C
A		0	N/C	N/C
B		7	7	7
CHS		5	54	54
CH	Försök med KH(5) och TCH(4)			
CK	Försök med K(5) och TSK(45)			
CZ, CS	CSZ, CZS	4	4	4
C	Försök med K(5) och TZ(4)			
DRZ, DRS		4	4	4
DS, DSH	DSZ	4	4	4
DZ DZH	DZS	4	4	4
D DT		3	3	3
EI EJ	EY	0	1	N/C
EU		1	1	N/C
E		0	N/C	N/C
FB		7	7	7
F		7	7	7
G		5	5	5
H		5	5	N/C
IA IE	IO IU	1	N/C	N/C
I		0	N/C	N/C
J	Försök med Y(1) och DZH(4)			
KS		5	54	54
KH		5	5	5
K		5	5	5
L		8	8	8
MN		66	66	66
M		6	6	6
NM		66	66	66

N		6	6	6
OI OJ	OY	0	1	N/C
O		0	N/C	N/C
P PF	PH	7	7	7
Q		5	5	5
RZ RS	Försök med RTZ(94) och ZH(4)			
R		9	9	9
SCHTSCH	SCHTSH SCHTCH	2	4	4
SCH		4	4	4
SHTCH SHCH	SHTSH	2	4	4
SHT SCHT	SCHD	2	43	43
SH		4	4	4
STCH STSCH	SC	2	4	4
STRZ STRS	STSH	2	4	4
ST		2	43	43
SZCZ SZCS		2	4	4
SZT SHD	SZD, SD	2	43	43
SZ		4	4	4
S		4	4	4
TCH TTCH	TTSCH	4	4	4
TH		3	3	3
TRZ TRS		4	4	4
TSCH TSH		4	4	4
S TTS	TTSZ, TC	4	4	4
TZ TTZ	TZS, TSZ, TS	4	4	4
T		3	3	3
UI UJ	UY	0	1	N/C
U UE		0	N/C	N/C
V		7	7	7
W		7	7	7
X		5	54	54
Y		1	N/C	N/C
ZDZ ZDZH	ZHDZH	2	4	4
ZD ZHD		2	43	43
ZH ZS	ZSCH, ZSH	4	4	4
Z		4	4	4

Bilaga 2

Phonix substitutionsparametrar

Numeriska teckenvärden:

1	*B, P*	5	*M, N*
2	*C, G, J, K, Q*	6	*R*
3	*D, T*	7	*F, V, Z*
4	*L*	8	*S, X*

Fonetiska substitutioner

(Sub byter ut till tecknen från början, mitten eller slutet av ordet)
Där v=vokal och c=konstant

Sub	Början	Mitten	Slut
G	DG	DG	DG
KO	CO	CO	CO
KA	CA	CA	CA
KU	CU	CU	CU
SI	CY	CY	CY
SI	CI	CI	CI
SE	CE	CE	CE
KL	CL om CLv		
K	CK	CK	CK
K			GC
K			JC
KR	CHR om CHRv		
KR	CR om CRv		
R	WR		
NK	NC	NC	NC
KT	CT	CT	CT
F	PH	PH	PH
AR	AA	AA	AA
SH	SCH	SCH	SCH
TL	BTL	BTL	BTL
T	GHT	GHT	GHT
ARF	AUGH	AUGH	AUGH
LD		LJ om vLJv	
LOW	LOUGH	LOUGH	LOUGH
KW	Q		
N	KN		
N			GN
N	GHN	GHN	GHN
N			GNE
NE	GHNE	GHNE	GHNE

NS			GNES
N	GN		
N		GN om GNc	GN om GNc
S	PS		
T	PT		
C	CZ		
Z		WZ om vVZ	
CH		CZ	
LSH	LZ	LZ	LZ
RSH	RZ	RZ	RZ
S		Z om Zv	
TS	ZZ	ZZ	ZZ
TS		Z om cZ	
REW	HROUG	HROUG	HROUG
OF	ROUGH	OUGH	OUGH
KW		Q om vQv	
Y		J om vJv	
Y	YJ om YJv		
G	GH		
E			GH om vGH
S	CY		
NKS	NX	NX	NX
F	PF		
T			DT
TIL			TL
DIL			DL
ITH	YTH	YTH	YTH
CH	TJ om TJv		
CH	TSJ om TSJv		
T	TS omTSv		
CHE	TCH	TCH	TCH
VSKIE		WSK om vWSK	WSK om vWSK
N	MN om MNv		
N	PN om PNv		
Sl		STL om vSTL	STL om vSTL
ENT			TNT
OH			EAUX
ECS	EXCI	EXCI	EXCI
ECS	X	X	X
ND			NED
DR	JR	JR	JR
EA			EE
S	ZS	ZS	ZS
AH		R om vRc	R om vRc

AH		HR om vHRc	HR om vHRc
AH			HR om vHR
AR			RE
AH			R om vR
LE	LLE	LLE	LLE
ILE			LE om cLE
ILES			LES om cLES
Tas bort			E
S			ES
AS			SS OM vSS
M			MB om vMB
MPS	MPTS	MPTS	MPTS
MS	MPS	MPS	MPS
MT	MPT	MPT	MPT

Referenser

Rahul Bhagat och Eduard Hovy, Phonetic Modells for Generating Spelling Variants, *IJCAI-07*, p 1570-1575, 2007

Randy Daitch, *AVOTAYNU*, Volume 2, Issue 1, p 19, 1986, http://www.avotaynu.com/soundex.html, (verifierad mars 2008)

T.N. Gadd: PHONIX: The Algorithm, Program: automated library and information system 24/4, 1990, p.363-366.

DDSS, http://www.ddss.nu/(S(ax2xsd45nmewxn45lb0xxrbk))/swedish/default.aspx, (verifierad mars 2008)

DigiArkiv, http://www.digiarkiv.se/, (länk verifierad mars 2008)

Donald Knuth, *The Art Of Computer Programming*, Volume 3, 1973, Addison-Wesley

Donald Knuth, *The Art of Computer Programming*, Volume 3, edition 2, 1998, Addison-Wesley

Gary Mokotoff, *AVOTAYNU*, Volume 1, Issue 1, p 19, 1985, http://www.avotaynu.com/soundex.html, (verifierad mars 2008)

National Archives, http://www.archives.gov/publications/general-info-leaflets/55.html, (länk verifierad mars 2008)

Lawrence Philips, Hanging on the Metaphone, *Computer Language*, Vol. 7, No. 12, p39, 1990.

Lawrence Philips, *C/C++ Users Journal*, Volume 18 Issue 6, June 2000, http://www.ddj.com/cpp/184401251?pgno=2, (verifierad mars 2008)

Robert C Russell, U.S. Patent 1.261,.67, 1918, http://www.pat2pdf.org/pat2pdf/foo.pl?number=1261167, (verifierad mars 2008)

Robert C Russell U.S. Patent 1.435.663, 1922, http://www.pat2pdf.org/pat2pdf/foo.pl?number=1435663, (verifierad mars 2008)

SCB, http://www.scb.se/templates/Product____30895.asp, (verifierad mars 2008)

Robert L Taft, *Name Search Techniques*, New York State Identification and Intelligence System, Special Report No. 1, Albany, New York, 1970

VC# 2008, http://www.microsoft.com/express/vcsharp/, (verifierad mars 2008)

Wikipedia Information Retrievel, http://en.wikipedia.org/wiki/Information_retrieval, (verifierad mars 2008)

Wikipedia Levensthein distance, Vladimir Levenshtein, 1965, http://en.wikipedia.org/wiki/Levenshtein_distance, (verifierad mars 2008)

D. Randall Wilson, *Name Standardization for Genealogical Record Linkage*, Family & Church History Department, The church of Jesus Christ of Latter-day Saints, 2005, http://www.fht.byu.edu/prev_workshops/workshop05/, (verifierad mars 2008)