DİZGİ EŞLEME ALGORİTMALARININ ALFABEYE BAĞLI ETKİNLİKLERİNİN ARAŞTIRILMASI

Aydın CARUS [1] Abdul Kadir ERSİN[2] Altan MESUT[3]

[1,2,3] Trakya Üniversitesi, Mühendislik-Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, Edirne

[1]e-posta: aydinc@trakya.edu.tr [2]e-posta: abdulkadirersin@trakya.edu.tr
[3]e-posta: altanmesut@trakya.edu.tr

Özetçe

Metin işleme konusunda dizgi eşleme önemli bir yere sahiptir. Dizgi eşlemeye yönelik farklı birçok algoritma geliştirilmiştir. Dizgi eşleme algoritmalarında; kullanılan alfabe, varsa algoritmanın ön işlem aşaması ve dizgiyi bulmak için yaptığı karşılaştırma (deneme) sayısı algoritmanın performansına etki eden önemli parametrelerdir. Bu çalışmada, dizgi eşleme algoritmalarının, kullanılan alfabeye ve dizgi uzunluğuna bağlı etkinlikleri, süre bakımından incelenmiş, elde edilen sonuçlar değerlendirilmiştir. Ayrıca algoritmaların, Türkçe ve İngilizce gibi farklı gramer yapısına sahip doğal dildeki etkinliklerinin nasıl değişiklik gösterdikleri araştırılmıştır.

1. Giriş

Her gün artan miktarda metinsel verileri de içeren çeşitli biçimlerde veri üretilmekte ve saklanmaktadır. İnternetin yoğun bir şekilde kullanılmaya başlanmasından sonra saklanmış bu verilere çok sayıda erişim gerçekleştirilmektedir. Bu veriler içinde gerektiğinde bir bilginin bulunması sorun olarak ortaya çıkmaktadır. Veri aramayı mümkün kılacak farklı yöntemler kullanılmakla birlikte, metinsel veriler üzerinde arama işlemlerini etkin olarak gerçekleştirmek için dizgi eşleme algoritmaları kullanılmaktadır. Birbirinden farklı yöntemler kullanarak daha etkin bir şekilde dizgiyi bulmayı amaçlayan birçok dizgi eşleme algoritması geliştirilmiştir. Dizgi eşleme algoritmaları metin düzenleyicilerde, internet arama motorlarında, genler üzerinde yapılan çalışmalar gibi birçok farklı alanda yaygın bir şekilde kullanılmaktadır. Farklı programlama yöntemlerinde ve bilgisayar bilimlerinin çeşitli alanlarında dizgi eşleme algoritmaları önem teşkil etmektedir.

Dizgi eşlemede; Σ ile temsil edilen σ adet sonlu sayıda karakterden oluşan alfabe hem $x=x[0, 1, \ldots, m\text{-}1]$ ile gösterilen uzunluğu m olan dizgi hem de uzunluğu n olan $y=y[0, 1, \ldots, n\text{-}1]$ ile temsil edilen metnin oluşturulmasında kullanılır. Dizgi eşleme, metin içinde aranan dizginin en az bir ve genellikle tüm bulunduğu konumları bulmayı kapsamaktadır.

Bu araştırmada, önceki çalışmalardan [1, 2] farklı olarak algoritmaların performanslarının alfabeye bağımlılığının incelenmesinin yanı sıra algoritma performanslarının farklı gramer yapısına sahip doğal dillere göre performanslarının farklılık gösterip göstermedikleri incelenmiştir. Bu amaçla, oluşturulmuş derlemler içinden belirlenmiş olan dizgiler, seçilen algoritmaları kullanan programlar ile aratılarak süreleri tespit edilmiştir. Tespit edilen bu süreler değerlendirilerek algoritmaların; alfabelere ve doğal dillere göre de farklı sonuçlar gösterip göstermediği ortaya koyulmaya çalışılmıştır.

2. Kullanılan Algoritmalar

Bu çalışmada farklı niteliklere sahip olan 13 adet algoritma seçilmiştir. Bu algoritmalar arama yaparken özelliklerine göre soldan sağa veya sağdan sola doğru belirli bir düzen içerisinde ya da düzensiz şekilde arama yapabilirler. Bu amaçla arama yöntemleri belirlenirken farklı arama yönüne sahip algoritmalar seçilmesine önem verilmiştir. Bazı algoritmalar dizgi arama için bir ya da birden fazla önişlem safhası kullanırken, bazıları ise hiçbir önişlem safhası kullanmayan algoritmalardır. Bu bölümde, incelenen 13 algoritma hakkında bilgi yer almaktadır.

Brute Force Algoritması: Bu algoritma programlanması ve anlaşılması kolay bir algoritmadır. Özel bellek gereksinimi yoktur. Hiçbir önişlem safhası gerektirmez. Karşılaştırma sonrası kaydırma her zaman sağa doğru 1 birimdir. Karşılaştırma düzensiz şekilde olabilmektedir. O(mn) zaman karmaşıklığına sahiptir. Beklenen karakter karşılaştırma sayısı $2n$'dir[1].

Morris-Pratt Algoritması: Bu algoritma Brute Force algoritmasının bir türevidir. Dizgi üzerinde önek ve soneklere göre bir kaydırma değeri hesaplar. Bu değer önişlem safhasında gerçekleştirilir. Eşitlik olmaması durumunda kaydırma, hesaplanan bu kaydırma değerine göre yapılır. Arama işlemini soldan sağa doğru gerçekleştirir. Önişlem safhasında zaman karmaşıklığı $O(m)$ iken arama safhasında ise $O(m+n)$ karmaşıklığındadır [3].

Knuth-Morris-Pratt Algoritması: Morris-Pratt algoritmasının iyi bir analizi sonrasında yapılan bir algoritmadır. Anlaşılması ve programlanması karmaşıktır. Morris-Pratt'ten farklı olarak sadece önişlem safhasında dizgi üzerinde aynı olan karakterleri de göz önünde tutarak bir kaydırma değeri hesaplar. Arama safhası Morris-Pratt algortiması ile aynıdır. Arama işlemine soldan sağa doğru gerçekleştirir. Zaman karmaşıklığı Morris-Pratt algoritması ile aynıdır [4].

Apostolico-Crochemore Algoritması: Knuth-Morris-Pratt algoritmasındaki önişlem safhasını kullanarak kaydırma işlemini gerçekleştirir. Aramayı soldan sağa doğru yapar. Ön işlem zaman karmaşıklığı O(m) iken ***a***rama safhasında zaman karmaşıklığı O($m+n$) olur [5].

Boyer-Moore Algoritması: Aramayı sağdan sola doğru gerçekleştirir. Arama dizgiden bir şeyler öğrenip olabildiğince ileri kaydırma yapma mantığına dayanır. Kötü karakter ve iyi sonek olmak üzere iki önişlem safhası bulunmaktadır.

[1] http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf

Kaydırma işlemlerini önişlem safhalarında elde edilen kaydırma değerlerinden maksimum olanını seçerek yapar. m dizgi uzunluğu, n metin uzunluğu ve Σ alfabe eleman sayısı olarak alındığında önişlem safhasında zaman karmaşıklığı O($m+\Sigma$) olur. Arama safhasında ise bu değer O(mn) dir [6].

Turbo Boyer-Moore Algoritması: Boyer-Moore algoritmasının bir türevidir. Boyer-Moore algoritmasındaki kötü karakter ve iyi sonek önişlem fazlarını kullanır. Bunun dışında ekstra bir önişlem safhası gerektirmez. Boyer-Moore algoritmasında olduğu gibi aramayı sağdan sola doğru gerçekleştirir. Boyer-Moore algoritmasına göre sabit olarak fazladan alana ihtiyacı vardır. Zaman karmaşıklığı Boyer-Moore algoritması ile aynıdır [7].

Horspool Algoritması: Boyer-Moore algoritmasının basitleştirilmiş halidir. Boyer-Moore algoritmasındaki kötü karakter önişlem fazını kullanır. Sağdan sola doğru karşılaştırma yapar. İlk olarak dizginin son karakterinden karşılaştırmaya başlanır. Eğer eşleme olmazsa kötü karakter kuralındaki kaydırma değeri kadar sağa kaydırılır. Eşleşme olması durumunda ise sola doğru karşılaştırmaya devam edilir. Önişlem safhasında zaman karmaşıklığı O($m+\Sigma$) , alan karmaşıklığı ise $O(\Sigma)$ dir. Arama safhasında ise bu değer O(mn) olur [8].

Quick Search Algoritması: Boyer-Moore algoritmasının basitleştirilmiş halidir. Boyer-Moore algoritmasındaki kötü karakter önişlem fazını kullanır. Horspool algoritmasının zaman ve alan karmaşıklığına sahiptir [9].

Tuned Boyer-Moore Algoritması: Boyer-Moore algoritmasının basitleştirilmiş halidir. Boyer-Moore algoritmasındaki kötü karakter önişlem fazını kullanır. Karşılaştırma düzensiz olarak yapılabilir. Bu algoritma en kötü durumda kuadratik zaman karmaşıklığı gösterirken arama davranışı en iyi derecededir. Kaydırma işlemini 3 aşamadan sonra gerçekleştirmektedir [10].

Zhu-Takaoka Algoritması: Boyer-Moore algoritmasının bir türevidir. Karşılaştırma işlemi sağdan sola doğru gerçekleştirilir. Boyer-Moore algoritmasındaki kötü karakter kaydırmasını hesaplamak için ardıl metin karakterlerini kullanır. İyi sonek önişlem safhası ise Boyer-Moore algoritmasındaki ile aynıdır. Kaydırma işlemini gerçekleştirirken bu iki kural arasındaki maksimum değerini seçerek kaydırma yapar. Önişlem safhasında zaman karmaşıklığı O($m+\Sigma^2$) olur. Arama safhasında ise bu değer O(mn) dir [11].

Berry-Ravindran Algoritması: Quick Search ve Zhu-Takaoka algoritmalarının birleşimi ile oluşturulmuş bir algoritmadır. Boyer-Moore algoritmasındaki kötü karakter kuralı bu iki algoritmanın birleşimi şekline dönüştürülmüştür. Bu kuraldan elde dilen değerler kaydırma değeri olarak kullanılmaktadır. Önişlem safhasında zaman ve alan karmaşıklığı O($m+\Sigma^2$) dir. Arama safhasında ise bu karmaşıklık O(mn) dir [12].

Smith Algoritması: Bu algoritma kaydırma yaparken Horspool algoritmasındaki kötü karakter kuralı ve Quick Search algoritmasındaki kötü karakter kuralı arasındaki maksimum değeri alır. Önişlem safhasında zaman karmaşıklığı O($m+\Sigma^2$) iken arama safhasında ise O(mn) dir. En kötü durumda zaman karmaşıklığı kuadratiktir [13].

Raita Algoritması: Bu algoritma karşılaştırma yaparken öncelikle son karakteri karşılaştırır. Eşleşme olması durumunda ise ilk karakteri karşılaştırır. Eşlemenin tekrar etmesi durumunda orta karakteri karşılaştırır. Eğer eşleşme devam ediyorsa karşılaştırmaya ikinci karakterden itibaren sona kadar yapar. Fakat önceden karşılaştırmış olduğu orta karakteri fazladan tekrar karşılaştırır. Kaydırma işlemi Horspool algoritmasındaki gibidir. Önişlem safhasında zaman karmaşıklığı O($m+\Sigma^2$) olur. Arama safhasında ise bu değer O(mn) olur. En kötü durumda zaman karmaşıklığı kuadratiktir [14].

3. Test Ortamı ve Derlemler

Dizgi eşleme algoritmalarının test edilmesinde kullanılmak üzere farklı gramer yapısına sahip diller olan Türkçe ve İngilizce için doğal dil ile yazılmış metinlerde kullanılan her biri için farklı 121 elemanı olan iki ayrı derlem oluşturulmuştur. Ayrıca A, C, G ve T elemanlarına sahip DNA alfabesi kullanılarak DNA derlemi ve 0-9 arası rakamlar kullanılarak rakam derlemi oluşturulmuştur. Metin uzunluğu dizgi eşleme algoritmalarını farklı yönde etkileyeceğinden tüm derlemler, uzunlukları 30,071 KB olarak sabit olacak şekilde hazırlanmıştır. Rakam ve DNA derlemleri rastgele olarak üretilmiştir. Türkçe için doğal dil işlemeye yönelik hazırlanmış ODTÜ Derleminin [15] içindeki doğal dile ait olmayan cümle işaretleme amaçlı XML kodları temizlenmiş ayrıca içerisine çeşitli konularda makale hikâye ve romanlardan oluşan eklemeler yapılarak Türkçe derlem oluşturulmuştur. İngilizce için oluşturan derlem Gutenberg[2] projesinden hikâye ve romanlar eklenerek elde edilmiştir.

Dizgi eşleme algoritmalarının dizgi uzunluğuna bağlı olarak performanslarını değerlendirebilmek için dizgi uzunlukları 10 karakter (kısa dizgi), 75 karakter (orta uzunluklu dizgi) ve 150 karakter (uzun dizgi) olarak seçilmiştir. Her bir derlem içerisinden kısa, orta ve uzun dizgi uzunlukları için metin içerisinde farklı konumlarda olacak şekilde 100 er adet dizgi tespit edilerek aramada kullanılacak dizgiler elde edilmiştir. Doğal diller için oluşturulan dizgiler doğal dile özgü anlamlı kelimeleri içerecek şekilde dizgi başı kelime ortasına gelmeyecek şekilde oluşturulmuştur.

Seçilen algoritmaların C programlama dili ile kodları yazılarak Visual Studio .NET 2005 ortamında *Release* türünde derlenmiş ve elde edilen programlar Windows işletim sistemi ile çalışan Core 2 Duo 1,83 Ghz İşlemci, 1024 MB ana bellek, 80 GB sabit diske sahip olan bilgisayar kullanılarak karşılaştırma sayıları ve süreleri elde edilmiştir.

4. Denemeler ve Sonuçlar

4.1. Doğal Dil Sonuçları

Hazırlanmış olan programları kullanılarak her bir dizgi uzunluğu için 100 dizgi arama işlemi gerçekleştirilmiştir. Bu bölümde verilen grafiklerdeki değerler 100 dizgi arama süresi değerlerini ifade etmektedir.

Doğal dil derlemleri üzerinde dizgi eşleme algoritmaları kullanılarak yapılan arama denemelerinden elde edilen arama süreleri incelendiğinde;

- Tablo 1'de görüldüğü gibi; tüm algoritmalarda kısa, orta ve uzun dizgiler için Türkçe derlem üzerinde yapılan

[2] http://www.gutenberg.org

aramalar, İngilizce derlem üzerinde yapılan aramalardan daha kısa sürmektedir. Süre bakımından; kısa ve uzun dizgiler için bazı algoritmalar yaklaşık %9,3 (Knut-Morris-Pratt algoritması) oranında daha hızlı olabilmektedir. Orta uzunluklu dizgilerde ise bazı algoritmalarda bu oran yaklaşık %13,5 (Turbo Boyer-Moore algoritması) oranına varmaktadır.

- Kısa dizgiler için Türkçe ve İngilizce derlemlerde en iyi süre performansına sahip olan algoritmalar Raita ve Horspool algoritmaları iken orta ve uzun dizgilerde en iyi arama süresine Zhu-Takaoka algoritması sahiptir. Tüm dizgi uzunluklarında en kötü performansı Apostolico-Crochemore algoritması vermiştir.

Tablo 1: Doğal diller için süre değerleri (s)

		TÜRKÇE			İNGİLİZCE		
		Kısa	Orta	Uzun	Kısa	Orta	Uzun
Yöntem	Horspool	2,40	1,34	1,47	2,59	1,48	1,48
	Raita	2,40	1,36	1,45	2,57	1,59	1,58
	Zhu-Takaoka	2,95	1,26	1,45	2,96	1,31	1,47
	Tuned Boyer-Moore	2,59	1,48	1,55	2,67	1,58	1,59
	Quick Search	2,75	1,54	1,58	3,13	1,55	1,63
	Berry-Ravindran	3,57	1,33	1,73	3,67	1,33	1,76
	Boyer-Moore	3,35	1,73	1,70	3,65	1,83	1,81
	Smith	3,90	1,70	1,65	4,42	1,78	1,72
	Turbo Boyer-Moore	4,15	1,79	1,77	4,60	2,07	1,88
	Brute Force	12,73	12,67	11,93	13,95	12,70	13,12
	Morris-Pratt	15,37	15,40	14,85	16,71	15,57	15,87
	Knuth-Morris Pratt	20,61	20,70	19,81	22,37	20,69	21,26
	Apostolico Crochemore	22,72	23,12	23,21	23,08	23,59	23,56

4.2. Alfabe Sonuçları

Bu değerlendirme aşamasında ise DNA, Rakam ve Doğal Dil olmak üzere 3 farklı alfabe seçilmiş ve algoritmaların bu 3 alfabe üzerindeki performansları dizgi uzunluğuna göre değerlendirilmiştir. Doğal Dil alfabesi için Türkçe seçilmiştir.

4.2.1. DNA Alfabesi Açısından

Tablo 2'de görüldüğü gibi; bazı algoritmaların DNA alfabesi üzerinde performansı dizgi uzunluğu arttıkça artarken bazılarında ise düşmektedir. Örneğin Turbo Boyer-Moore algoritmasının kısa dizgiden uzun dizgiye geçişte performansı %59 oranında artarken, Tuned Boyer-Moore algoritmasının performansı kısa dizgiden uzun dizgiye geçişte %62 oranında azalmaktadır. Apostolico-Crochemore, Morris-Pratt ve Knuth-Morris-Pratt algoritmaları da kısa dizgilerden uzun dizgilere geçişte yaklaşık %80 oranlarında performans artışı göstermektedirler. Boyer-Moore türevi algoritmaların genelinde DNA alfabesi üzerinde dizgi uzunluğu arttıkça performansın da azaldığı görülebilmektedir.

4.2.2. Rakam Alfabesi Açısından

Bu alfabe üzerinde ise Apostolico-Chrochemore, Morris-Pratt ve Knuth-Morris-Pratt algoritmaları DNA alfabesinde olduğu gibi kısa dizgilerden uzun dizgilere geçişte performans artışı sağlamaktadır. Fakat bu performans artışı DNA alfabesinde olduğu kadar etkili değildir. Bu durum bu algoritmaların alfabe uzunluğu arttıkça performans düşüklüğü meydana getirdiklerinin açık bir kanıtıdır.

Tuned Boyer-Moore algoritması da DNA alfabesinde olduğu gibi kısa dizgilerden uzun dizgilere geçişte %62 oranında performans düşüklüğü göstermektedir. Bu performans düşüklüğü Raita algoritmasında %50 iken, Zhu-Takaoka algoritmasında %29 oranındadır. Turbo Boyer-Moore ise kısa dizgiden orta uzunluklu dizgiye geçişte performans artışı sağlarken bunu uzun dizgiye geçişte koruyamamaktadır. Bu performans düşüklüğünün ana sebebi ise uzun dizgilerin önişlem safhasında işlenirken zaman harcamasıdır.

4.2.3. Doğal Dil Alfabesi Açısından

Bu alfabe üzerinde ise performans değişiklikleri Boyer-Moore türevleri üzerinde daha etkin bir biçimde görülebilmektedir. Kısa dizgilerden orta uzunluklu dizgilere geçişte yaklaşık %63 oranında performans artışı görülmektedir. Bu değeri Berry-Ravindran algoritması sağlarken, Zhu-Takaoka, Smith ve Turbo Boyer-Moore algoritmaları %58 oranlarında performans artışı sağlamaktadır. Bu oran Boyer-Moore, Raita, Horspool ve Quick Search algoritmalarında yaklaşık %47 oranında iken, Tuned Boyer-Moore algoritmasında ise %43 oranındadır. Orta uzunluklu dizgilerden uzun dizgilere geçişte

ise genel olarak yaklaşık %14 oranında bir performans düşüklüğü meydana gelmektedir. Bu sonucu daha önceden de belirttiğimiz şekilde önişlem safhasındaki dizgi işleme süresi oluşturmaktadır. Brute Force türevi tüm algoritmalar ise tüm dizgi uzunluklarında kendi içlerinde çok yakın sonuçlara sahiptirler.

Tablo 2: Alfabeler için süre değerleri (s)

		DNA			RAKAM			DOĞAL DİL		
		Kısa	Orta	Uzun	Kısa	Orta	Uzun	Kısa	Orta	Uzun
Yöntem	Zhu-Takaoka	7,66	5,82	7,99	4,21	3,00	4,24	2,95	1,26	1,45
	Berry Ravindran	8,89	7,79	10,72	4,79	2,82	3,79	3,57	1,33	1,73
	Horspool	9,33	11,93	14,26	4,09	3,84	4,87	2,40	1,34	1,47
	Boyer Moore	11,39	8,10	11,78	5,76	4,93	6,36	3,35	1,73	1,70
	Turbo Boyer Moore	13,49	9,02	7,96	7,47	5,62	6,86	4,15	1,79	1,77
	Tuned Boyer Moore	7,89	13,79	20,58	3,20	5,07	8,35	2,59	1,48	1,55
	Raita	9,39	14,88	20,80	4,24	5,55	8,36	2,40	1,36	1,45
	Smith	13,06	16,02	19,36	5,44	4,45	6,43	3,90	1,70	1,65
	Quick Search	14,66	17,46	20,92	5,16	4,87	6,65	2,75	1,54	1,58
	Morris Pratt	28,05	16,18	10,75	19,76	17,35	15,21	15,37	15,40	14,85
	Apostolico-Crochemore	25,33	11,89	5,99	24,84	19,48	13,21	22,72	23,12	23,21
	Brute Force	30,51	33,68	36,66	17,07	18,63	21,46	12,73	12,67	11,93
	Knuth Morris Pratt	32,59	21,28	15,82	24,94	22,43	20,17	20,61	20,70	19,81

5. Sonuç

Algoritmaların performansları alfabeye ve dizgi uzunluğuna bağlı olarak değişiklik göstermektedir. Alfabedeki eleman sayısı azaldıkça genellikle algoritmaların performansı da düşmektedir. Dizgi uzunluğu alfabelerin performansını etkilemektedir. Algoritmaların performanslarının alfabeye ve dizgi uzunluğuna bağlı olmasının yanı sıra, aynı alfabe eleman sayısına sahip olan doğal dillerde dilin yapısından kaynaklanan bazı etkilenmeler olabilmekte ve bunun sonucunda doğal diller arasında arama performans farklılıkları doğabilmektedir. Yapılan bu çalışmada birçok algoritmanın hangi alfabeler üzerinde ne kadar etkili olduğu ve birbirlerine göre performans durumları açıkça gösterilmiş ve Türkçe dil yapısının İngilizce dil yapısına göre çoğu algoritmada daha etkin sonuçlar verdiği ortaya konmuştur. Arama algoritmalarının dünyada kullanılan diğer doğal dillere göre nasıl farklılıklar gösterdiği ise bundan sonraki çalışmada ele alınacaktır.

Kaynakça

[1] Lecroq, T., "Experimental results on string matching algorithms", *Software - Practice & Experience*, 25(7):727-765, 1995.

[2] Lecroq, T., "New experimental results on exact string-matching", LIFAR-ABISS, Facultédes Sciences et Techniques, Université de Rouen, 2000.

[3] Morris (Jr), J. H., Pratt, V. R., "A linear pattern-matching algorithm", Technical Report 40, University of California, Berkeley, 1970.

[4] Knuth, D. E., Morris (Jr), J. H., Pratt V. R., "Fast pattern matching in strings", *SIAM Journal on Computing* 6(1):323-350, 1977.

[5] Apostolico, A., Crochemore M., "Optimal canonization of all substrings of a string", *Information and Computation* 95(1):76-95, 1991.

[6] Boyer, R. S., Moore, J. S., "A fast string searching algorithm". *Communications of the ACM*. 20:762-772, 1977.

[7] Crochemore, M., Czumaj, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W., "Deux méthodes pour accélérer l'algorithme de Boyer-Moore", in *Théorie des Automates et Applications, Actes des 2e Journées Franco-Belges*, D. Krob ed., Rouen, France, 45-63, 1991.

[8] Horspool, R. N., "Practical fast searching in strings", *Software - Practice & Experience*, 10(6):501-506, 1980.

[9] Sunday, D. M., "A very fast substring search algorithm", *Communications of the ACM*, 33(8):132-142, 1990.

[10] Hume, A., Sunday, D. M., "Fast string searching", *Software - Practice & Experience* 21(11):1221-1248, 1991.

[11] Zhu, R. F., Takaoka, T., "On improving the average case of the Boyer-Moore string matching algorithm", *Journal of Information Processing*, 10(3):173-177, 1987.

[12] Berry, T., Ravindran, S., "A fast string matching algorithm and experimental results", in *Proceedings of the Prague Stringology Club Workshop`99*, J. Holub and M. Simánek ed., Collaborative Report DC-99-05, Czech Technical University, Prague, Czech Republic, 1999,16-26.

[13] Smith, P. D., “Experiments with a very fast substring search algorithm”, *Software - Practice & Experience* 21(10):1065-1074, 1991.

[14] Raita, T., “Tuning the Boyer-Moore-Horspool string searching algorithm”, *Software - Practice & Experience*, 22(10):879-884, 1992.

[15] Say, B., Zeyrek, D., Oflazer, K., Özge, U., “Development of a Corpus and a Treebank for Present-day Written Turkish”, *Proceedings of the Eleventh International Conference of Turkish Linguistics*, 183-192, 2002.