The history of finding palindromes

Johan Jeuring

Technical Report UU-CS-2013-008

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

The history of finding palindromes

Johan Jeuring[1,2]

[1] Department of Information and Computing Sciences, Universiteit Utrecht
[2] School of Computer Science, Open Universiteit Nederland
P.O.Box 2960, 6401 DL Heerlen, The Netherlands
J.T.Jeuring@uu.nl

Abstract. This paper describes the history of finding palindromes in computer science. The problem of determining whether or not a string is a palindrome is one of the oldest computer science problems, and algorithms for this problem have been constructed since the early years of computer science. This paper describes the contributions to solving algorithmic problems related to finding palindromes and variants of palindromes.

1 Introduction

Fig. 1. The Sator square, from a Swedish manuscript, 1722, Skara, Sweden [34]

Since 1998 I have worked in the group "SDS", the name used for the Software Technology group in internal documents of the department of Information and Computing Sciences of Utrecht University. The group is called after the initials of Doaitse Swierstra. The most remarkable aspect of these initials is that they constitute a *palindrome*. Until a successor of Doaitse has been appointed, I will

be group leader of the Software Technology group, and, preserving all properties, its name should probably change to "JTJ".

Doaitse thinks palindromes are boring. He and I wrote a set of lecture notes together on 'Languages and compilers' [23], which uses palindromes as one of its first examples. One of Doaitse's returning remarks about these notes is that it is so boring to use palindromes as an example. In this paper I show that palindromes have a rich history, also in computer science, and that they are a worthwhile object of study. I will refer to many books and papers about formal languages, computing models, and algorithms, in which palindromes are used as first or second example, or as inspiration for other results. The books and papers in the list of references of this paper have been cited almost 40.000 times, and this is probably my paper with the most cited citations. I doubt it will be sufficient for Doaitse to change his mind, however. Since Doaitse will hopefully not be the only reader of this paper, I will also include some descriptions of concepts of which Doaitse is well aware, so that readers with a different computer science background can also learn something about the history of palindromes in computer science.

The palindrome concept has a long history. Already around 2000 years ago, the palindrome 'Rotas opera tenet arepo sator', see also Figure 1, was written on the walls of Pompeii. Its precise meaning is unclear. Some 300 years before that, around 250 BC, Sotades reportedly wrote the first verses that also made sense if read backwards. Unfortunately, none of the palindromic verses of Sotades survived.

The oldest Dutch palindrome I could find is 'Neder sit wort trow tis reden', from 1584, see Figure 2. The meaning of this sentence is unclear too, one possibility is 'Humility is the word, loyal the intellect'

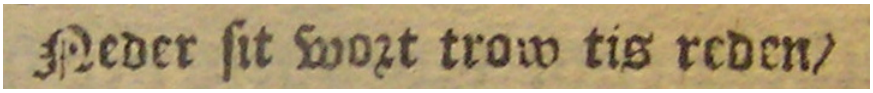

Fig. 2. Dutch palindrome ('letter-kreeftdicht') from 1584 [33]

Palindromes have long been considered interesting curiosities used in word-plays. We now know that palindromes play an important role in DNA. If I search for the keyword palindrome in the electronic publications available at the library of Utrecht University, I get more than 500 hits. The first ten of these hits are all about palindromes in DNA. My guess is that at least 90% of these 500 publications are about palindromes in DNA. For example, the male DNA contains huge approximate palindromes with gaps in the middle [30]. Some of these palindromes are more than a million base-pairs long. The genes for male testes are encoded on one of these palindromes. The male chromosome, XY, consists of a single X and a single Y, and is the only chromosome that does not have two

copies of the same DNA string. An important reason why chromosomes have two copies of the same string is to repair possible errors in DNA. Since the Y chromosome lacks a copy, it needs to resort to other mechanisms for repairing itself. The Y chromosome uses palindromes for this purpose: the important genetical information in Y appears in palindromes. Thus palindromes play an important role in saving males from extinction...

We need software to find palindromes in large pieces of text, or approximate palindromes with gaps in DNA. Algorithms for determining whether or not a string is a palindrome, and finding palindromes in strings have a long history in computer science, longer than Doaitse's career. On the occasion of Doaitse's retirement I want to look back to a very small part of the history of computer science, and revisit the history of algorithms for finding palindromes.

2 Finding palindromes

2.1 Formal language theory

Palindromes have been used as examples in formal language theory, and to illustrate the power of various computing models.

The classic book of Noam Chomsky introducing syntactic structures for languages from 1957 [6] uses palindromes as its second example. The first example of a formal language in Hopcroft and Ullman's Introduction to Automata Theory, Languages and Computation [20] is the language of palindromes. A predecessor of this book [19] shows that the language of even-length palindromes is a nondeterministic context-free language: a context-free language that can be recognized by means of a pushdown automaton, but not by a deterministic pushdown automaton. This implies that this language is context-free, but 'of a more involved' kind.

Stephen Cole showed how to recognize palindromes on iterative arrays of finite-state machines in 1964 [7,8], and so did Seiferas on iterative arrays with direct central control [29]. Alvy Ray Smith III used cellular automata to recognize palindromes in 1971 [32]. Also in the 1970s, Fréjvald [13] and Yao [36], independently, looked at recognizing palindromes by means of probabilistic Turing machines. In 1965, Fréjvald [12] and Barzdins [3] independently showed that it requires a number of steps quadratic in the length of the input string on a Turing machine with a single head and a single tape to determine whether or not a string is a palindrome. In his review of these papers, Mullin [27] conjectured that this problem can be solved in linear time on a machine with two heads. Eight years later in 1973, Slisenko [31] showed that if you are allowed to use more than one head on a tape, or multiple tapes, then indeed a palindrome can be determined in a number of steps linear in the length of the input string. Looking at these results from a distance of around 40 years, it seems like in those days it was a sport to study programming with various restrictions, like not using your left hand, or tying your legs together.

The English translation of Slisenko's paper is a 183 page, dense mathematical text. Slisenko announced his result already in 1969, and given the form and

the length of his paper (with 183 pages this is more a book than a paper), I find it highly likely that it took him a couple of years to write it. Slisenko's result was a surprisingly strong result, and people started to study and trying to improve upon it. One of these people was Zvi Galil, then a postdoc IBM Yorktown Heights. He had a hard time understanding Slisenko's paper, which I fully understand, but in 1978 he obtained the same result, only he needed just 18 pages to describe it. The problem of finding palindromes efficiently started off an area within computer science now known as stringology, which studies strings and their properties, and algorithms on strings.

2.2 Stringology

A nice example of how palindromes influenced the development of well-known algorithms is given by Knuth in Knuth, Morris and Pratt's paper about fast pattern matching in strings [24]. Daniel Chester had developed a program to recognize strings beginning with an even-length palindrome using a two-way deterministic pushdown automaton. Knuth had just learned about Cook's theorem, which stated that any language recognizable by a two-way deterministic pushdown automaton can be recognized in linear time on a Random Access Machine (RAM) [9]. After he had successfully applied Cook's procedure to obtain a linear-time RAM algorithm for finding even-length palindromes at the start of a string, he realised that he could use a similar procedure to obtain a fast algorithm for pattern matching. This algorithm later became know as the Knuth, Morris, Pratt (KMP) pattern matching algorithm. Around the same time, and in a similar way by moving between different computation models, Galil found a linear-time algorithm for finding initial palindromes of any length [15].

Some of the machine models on which algorithms for palindromes were developed have rather artificial restrictions, which gives rather artificial algorithms for finding palindromes. But some of the algorithms on the more realistic machine models, such as the RAM model, contain the essential components of a by now relatively well-known linear-time algorithm for finding palindromes. Manacher [26] gives a linear-time algorithm on the RAM computing model finding the smallest initial palindrome of even length. The difference with the algorithms described above is that this algorithm is 'on-line': it finds palindromes as it reads input symbols, and it doesn't need to see the complete input string. He also describes how to adjust his algorithm in order to find the smallest initial palindrome of odd length longer than 3. He did not realize that his approach could be used to find all maximal palindromes in a string in linear time. Zvi Galil and Joel Seiferas did, in 1976. They wrote a paper, titled 'Recognizing certain repetitions and reversals within strings' [17], in which they develop an algorithm that finds all maximal palindromes in a string in linear time. As far as I know, this is the first description of this algorithm.

Twelve years later I rediscovered this algorithm. I published a paper on 'The derivation of on-line algorithms, with an application to finding palindromes' [22], in which I show how to obtain the efficient algorithm for finding palindromes from the naive algorithm for finding palindromes using algebraic reasoning. The

method at which I arrive at the algorithm is completely different from the way Galil and Seiferas present their version. I presented the algorithm and the method I use to construct it to several audiences. I was only 22 years old at the time, and I am afraid my presentation skills were limited. Several people that saw me presenting the efficient algorithm for finding palindromes thought they could do better. An example can be found in the book 'Beauty is our business' (which isn't about models, or escort services, but about computer science, and dedicated to the Dutch Turing award winning computer scientist Edsger Dijkstra on his sixtieth birthday), in which Jan Tijmen Udding derives the same algorithm using Dijkstra's calculus for calculating programs [35].

The stringology community went on to develop algorithms with even more refined features for finding palindromes, such as a *real-time* algorithm, an algorithm that finds all palindromes in a string, but only uses a constant amount of time after reading each input symbol [14].

2.3 Finding palindromes in parallel

Zvi Galil developed algorithms for almost all problems related to palindromes in his series of papers on palindromes. In 1985, he developed an algorithm for finding initial, even-length, palindromes using a parallel machine [16] in time logarithmic in the length of the input string. Turing machines, and the other machine models mentioned above, are sequential machines: computations are performed in sequence, and not at the same time. Parallel machines allow computations to be performed at the same time, in parallel. In the previous century few parallel machines were available, but nowadays, even the laptop on which I am typing this text has four cores, and can do many things in parallel. The importance of parallel machines has increased considerably, also because it is expected that increasing the speed of computers by increasing the clock speed is not going to work anymore in the next couple of years. Extra power of computers has to come from the possibilities offered by using multiple cores for computations. To give an example: I searched for palindromes in the human Y chromosome using the efficient, linear-time, on-line algorithm. Since this chromosome has more than 20 million base-pairs, it takes quite some time (in the order of minutes) to find palindromes in it.

To put multiple cores to good use, we need efficient parallel algorithms for the problems we want to solve. Apostolico, Breslauer, and Galil improved upon Galil's first parallel algorithm for finding palindromes in their paper on 'Optimal Parallel Algorithms for Periods, Palindromes and Squares' in 1992 [1], based on an approach to use overhanging occurrences of strings to find initial palindromes introduced by Fischer and Paterson [11]. This optimal algorithm takes $log\ (log\ n)$ time using $n\ (log\ n)$ processors, where n is the length of the input string. The number of processors available to a user is usually fixed, and does not depend on the length of an input string. My machine has four cores, and it is impossible to change that number. Given the number of available processors, Breslauer and Galil determine how much time a parallel algorithm takes to find initial palindromes [4].

There are quite a few different different parallel architectures, depending on for example whether or not two or more processors can read or write a symbol from memory simultaneously. If we want to use one of these architectures to find palindromes, we need an algorithm specifically developed for this architecture, and we can construct different algorithms that optimize time, space, or use a particular number of processors. Further algorithms for finding palindromes on various parallel architectures have been developed by Crochemore and Rytter [10] and, again, Apostolico, Breslauer, and Galil [2].

2.4 Gapped and approximate palindromes

Since the discovery that DNA contains many palindromes, people working on bioinformatics have developed algorithms for finding palindromes in DNA. The first description of these algorithms I could find are in the book Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, by Dan Gusfield [18]. This is one of the standard works in computational biology. Gusfield shows how to compute all maximal palindromes in a string using a suffix tree. He then moves on to discuss gapped palindromes (called separate palindromes in Gusfield's book), and approximate palindromes. A gapped palindrome is a palindrome with a gap in the middle, and an approximate palindrome is a palindrome after a limited number of symbols is changed, deleted or inserted. These kinds of palindromes are important in computational biology, because, for example, the very long palindromes in the Y chromosome all have gaps, and a limited number of 'errors'. Gusfield leaves it to the reader to construct an algorithm that returns approximate palindromes in which only symbols can be changed (not deleted or inserted) in time linear in the product of the maximum number of errors and the length of the input.

After Gusfield, several other scientist have worked on finding gapped and approximate palindromes, sometimes improving on or generalizing Gusfield's results. For example, Porto and Barbosa have developed an efficient algorithm that finds approximate palindromes in which also symbols may be inserted or deleted [28], Kolpakov and Kucherov have developed several algorithms for determining palindromes with gaps [25], and Hsu, Chen, and Chao find all approximate gapped palindromes [21].

3 Conclusions

This short paper shows the history of describing and finding palindromes in computer science by discussing the literature on this topic. I have not included all literature about variants of the problem of finding palindromes: the number of variants is substantial, and listing the literature about all variants is infeasible. For example, I recently came across a paper that shows how to find approximate palindromes in run-length encoded strings [5].

I expect we will see solutions to more variants of the palindrome problem in the future. For example, the papers about parallel algorithms for finding

palindromes listed in this paper all describe algorithms for finding exact palindromes. Parallel algorithms are particularly useful for huge input strings, such as the human Y chromosome. The palindromes occurring in the Y chromosome are gapped and approximate, so we need to develop variants of the parallel algorithms for gapped approximate palindromes. Also from an algorithm-design perspective I think there are still some open questions. The central concept in the design of the efficient algorithm for finding palindromes is the palindromes in palindromes property. This property says that if a large palindrome contains a smaller palindrome that does not appear exactly in the middle, the large palindrome contains a second copy of the smaller palindrome at the other arm of the large palindrome. I think this property should also play a central role in efficient algorithms for finding gapped and approximate palindromes. I have tried for quite a while to design algorithms for these problems using the palindromes in palindromes concept, but failed. I hope someone else will solve this problem.

References

1. Alberto Apostolico, Dany Breslauer, and Zvi Galil. Optimal parallel algorithms for periods, palindromes and squares (extended abstract). In *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 296–307. Springer, 1992.
2. Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(1 - 2):163–173, 1995.
3. A.M. Barzdin. Complexity of recognition of symmetry on Turing machines (in Russian). *Problémy kibérnétiki*, 15:245–248, 1965.
4. Dany Breslauer and Zvi Galil. Finding all periods and initial palindromes of a string in parallel. *Algorithmica*, 14(4):355–366, 1995.
5. Kuan-Yu Chen, Ping-Hui Hsu, and Kun-Mao Chao. Efficient retrieval of approximate palindromes in a run-length encoded string. *Theoretical Computer Science*, 432(0):28–37, 2012.
6. Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.
7. Stephen N. Cole. *Real-time computation by iterative arrays of finite-state machines*. PhD thesis, Harvard University, 1964.
8. Stephen N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Transactions on Computers*, C-18(4):349–365, 1969.
9. Stephen A. Cook. Linear time simulation of deterministic two-way pushdown automata. In *IFIP Congress (1)*, pages 75–80, 1971.
10. Maxime Crochemore and Wojciech Rytter. Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. *Theoretical Computer Science*, 88(1):59–82, 1991.
11. Michael J. Fischer and Michael S. Paterson. String matching and other products. In R.M. Karp, editor, *SIAM AMS Proceedings on Complexity of Computation*, volume 7, pages 113–126, 1974.
12. R. Fréjvald. Complexity of recognition of symmetry on Turing machines with input (in Russian). *Algébra i logika, Séminar*, 4(1):47–58, 1965.
13. R. Fréjvald. Fast computation by probabilistic Turing machines. In *Theory of Algorithms and Programs*, volume 2, pages 201–205. Latvian State University, 1975.
14. Zvi Galil. Real-time algorithms for string-matching and palindrome recognition. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, STOC '76, pages 161–173. ACM, 1976.

15. Zvi Galil. Two fast simulations which imply some fast string matching and palindrome-recognition algorithms. *Information Processing Letters*, 4:85–87, 1976.
16. Zvi Galil. Optimal parallel algorithms for string matching. *Information and Control*, 67(1-3):144–157, 1986.
17. Zvi Galil and Joel I. Seiferas. Recognizing certain repetitions and reversals within strings. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 236–252, 1976.
18. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.
19. John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, 1969.
20. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
21. Ping-Hui Hsu, Kuan-Yu Chen, and Kun-Mao Chao. Finding all approximate gapped palindromes. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, volume 5878 of *ISAAC '09*, pages 1084–1093. Springer-Verlag, 2009.
22. Johan Jeuring. The derivation of on-line algorithms, with an application to finding palindromes. *Algorithmica*, 11:146–184, 1994.
23. Johan Jeuring and Doaitse Swierstra. *Lecture notes on Languages and Compilers*. Utrecht University, Department of Information and Computing Sciences, 2000.
24. Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1978.
25. Roman Kolpakov and Gregory Kucherov. Searching for gapped palindromes. *Theoretical Computer Science*, 410(51):5365–5373, 2009.
26. Glenn Manacher. A new linear-time 'on-line' algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22:346–351, 1975.
27. Albert A. Mullin. Review of Fréjvald [12] and Barzdins [3]. *The Journal of Symbolic Logic*, 35(1):159, 1970.
28. Alexandre H.L. Porto and Valmir C. Barbosa. Finding approximate palindromes in strings. *Pattern Recognition*, 35(11):2581–2591, 2002.
29. Joel I. Seiferas. Iterative arrays with direct central control. *Acta Informatica*, 8:177–192, 1977.
30. Helen Skaletsky, Tomoko Kuroda-Kawaguchi, Patrick J. Minx, Holland S. Cordum, LaDeana Hillier, Laura G. Brown, Sjoerd Repping, Tatyana Pyntikova, Johar Ali, Tamberlyn Bieri, Asif Chinwalla, Andrew Delehaunty, Kim Delehaunty, Hui Du, Ginger Fewell, Lucinda Fulton, Robert Fulton, Tina Graves, Shun-Fang Hou, Philip Latrielle, Shawn Leonard, Elaine Mardis, Rachel Maupin, John McPherson, Tracie Miner, William Nash, Christine Nguyen, Philip Ozersky, Kymberlie Pepin, Susan Rock, Tracy Rohlfing, Kelsi Scott, Brian Schultz, Cindy Strong, Aye Tin-Wollam, Shiaw-Pyng Yang, Robert H. Waterston, Richard K. Wilson, Steve Rozen, and David C. Page. The male-specific region of the human y chromosome is a mosaic of discrete sequence classes. *Nature*, 423(6942):825–837, 2003.
31. A.O. Slisenko. Recognizing a symmetry predicate by multihead Turing machines with input. In V.P. Orverkov and N.A. Sonin, editors, *Proc. of the Steklov Institue of Mathematics*, volume 129, pages 25–208, 1973.
32. Alvy Ray III Smith. Cellular automata complexity trade-offs. *Information and Control*, 18:466–482, 1971.
33. Hendrick Laurensz. Spieghel. *Twe-spraack vande Nederduitsche letterkunst*. Cristoffel Plantyn, Leiden, 1584.

34. Petter Jean Udd, Christian Papke, Petrus Sommar, and Tycho Brahe. *Samlingshandskrift med blandat, företrädesvis historiskt och praktiskt, innehåll.* 1765.
35. Jan Tijmen Udding. The maximum length of a palindrome in a sequence. In David Gries, editor, *Beauty is our business*, pages 410–416. Springer-Verlag New York, Inc., New York, NY, USA, 1990.
36. A.C. Yao. A lower bound to palindrome recognition by probabilistic Turing machines. Technical Report STAN-CS-77-647, Computer Science Department, Stanford University, 1977.