Sock Sorting

Laxmi Parida (IBM and NYU), Rohit Parikh (CUNY and NYU)
and Vaughan Pratt (Stanford)

April 1, 1999

Abstract

Algorithms of a social nature are extremely important and we are very dependent on certain "data structures" like names and social security numbers, on courts and on the postal system, and on various other social structures which enable us to perform activities like calling someone (you need her name for that), suing someone, or sending someone a bill. Unlike the situation in the theory of computation the existence of an algorithm for a real life situation always implies that the problem is feasibly solvable. Perhaps the closest to the unsolvability of the halting problem are Arrow's [1] impossibility theorems for voting. In this paper we analyze a rather mundane problem the problem of sorting socks after they have come out of the dryer and give several polynomial time algorithms

Contents

1 Introduction

Since Knuth [6] first wrote about toilet paper usage, a tradition has begun that theoretical computer scientists need not confine themselves to the analysis of computer algorithms but may also consider more mundane algorithmic situations.

Actually, such situations are not all that uncommon. We have many such algorithms, cooking recipes for example, or algorithms for knitting a sweater or for toilet training a puppy. But there is very little by way of a *theory* of such algorithms. The whole territory of ordinary life algorithms is relatively unexplored from a theoretical point of view. Thus when problems from ordinary life do arise which are capable of an exact mathematical treatment, then it is pleasant to consider them.

Algorithms of a social nature are extremely important and we are very dependent on certain "data structures" like names and social security numbers, on courts and on the postal system, and on various other social structures which enable us to perform activities like calling someone (you need her name for that), suing someone, or sending someone a bill. Unlike the situation in the theory of computation the existence of an algorithm for a real life situation always implies that the problem is feasibly solvable. Perhaps the closest to the unsolvability of the halting problem are Arrow's [1] impossibility theorems for voting.

Wittgenstein's language games [10] are a way of understanding certain philosophical puzzles through an investigation of such algorithmic issues embedded in the structure of society. See [9] for a fairly precise analogy between programming languages and natural language which exploits this sort of insight.

However, the issue before us right now is not a life and death matter for anyone and we look at it partly for amusement, partly also to illustrate the use of mathematical techniques in one more ordinary life problem.

In Knuth's puzzle, people sitting down in a stall in a public toilet face two rolls of toilet paper. Suppose that each person randomly selects k sheets from one of the two rolls of toilet paper. Knuth wants to know, when one of the two rolls becomes empty, how many sheets will still remain (on the average) on the non-empty roll.

Our own puzzle is 'cleaner' in that it does not rely on a public toilet but on a laundry room. In our puzzle, Ravi owns several pairs of socks and other clothes, and does his laundry sporadically. It may happen when he does his wash that he has in the washtub up to twenty five pairs of socks, along with other non-sock apparel. Ravi wears only dark gray or dark blue or black socks, each of various shades, and of course they are in matched pairs when he puts them in. After all they were in matched pairs when he wore them!

Alas, they are no longer this way when they come out of the wash and are then dried. What he now has is fifty individual socks, and not twenty five pairs. He now has to match them up using visual inspection, and this is where he runs into trouble. The trouble is that socks which are not precisely similar in color may still appear to the naked eye to match. Unfortunately, the binary relation of color indiscriminability is not transitive. It is possible for the following to hapen.

Say that socks s and s' are close enough in color so they *look* the same to Ravi even though their color is not exactly the same. Similarly for socks s' and s''. However, due to the fact that color indiscriminability is intransitive [2, 7], it may happen that socks s and s'' do not match. Thus suppose that he originally had three pairs $\{s, s\}, \{s', s'\}$ and $\{s'', s''\}$ and foolishly matched one of the two s' socks with an s and the other s' sock with an s''. He will now be left with socks s and s'' and these will no longer match!

Well, what is backtracking for? All he has to do is to toss the six socks (if that is all he had) into a pile and start over. Sooner or later he will get it.

But this is not quite so easy if he has, as we said, fifty socks and not merely six. There are $50!/(25! \times 2^{25})$ ways of matching (or mismatching) 50 socks into 25 pairs and this number is far too great. Is Ravi facing an NP-complete problem? Should he just throw the socks away and go buy twenty five brand new pairs rather than spend the rest of his life sorting?

To deal with this problem in a precise way we need to formulate a mathematical problem which is equivalent to the sock sorting problem.

2 Problem Formulation:

Construct a graph $\mathcal{G}$ where every vertex v_i corresponds to a sock s_i, $1 \leq i \leq n$, where n is the total number of socks. Introduce an edge $v_i v_j$ if socks v_i and v_j are indistinguishable to the naked eye. Call this the *indistinguishability graph*. In the sock sorting problem, the task is to find a subgraph $\mathcal{G}'$ of $\mathcal{G}$ whose vertex set is the same as that of $\mathcal{G}$ and the degree of every vertex is exactly one. Sock sorting (SS) is a special case of the *perfect matching* problem [3], in the sense that in an instance of the SS problem, the total number of vertices is always even and the degree of every vertex is odd. The perfect matching problem can be solved exactly in polynomial time, specifically in $O(n^{5/2})$ time [4]. However, the algorithm is not at all straightforward.

So Ravi *can* indeed sort his socks in polynomial time, though apparently not in linear time and the algorithm is not one that he can easily remember unless he is a theoretical computer scientist. Is there then a simple linear time algorithm which is easy to remember and which will give him, say, twenty pairs of matched socks, the rest to be tossed into a dustbin? What is the situation with a general n?

3 The naive (greedy) algorithm:

In the naive algorithm Ravi picks up one sock and tries to find a match among the unsorted socks. If he finds one, then the two constitute a pair. He puts them aside, and picks up another sock. If he does not find a match, he throws away the sock he had picked up, and starts with another sock.

Clearly the naive algorithm runs in $O(n^2)$ time. For each sock that he picks up, there will be at most n comparisons and there are n socks, so $O(n^2)$ comparisons in all. Also there is little storage requirement. At any time, there is (a) the sock s which is being paired at the moment, (b) the socks which have already been compared with s, and found not to match, (c) the socks which have yet to be compared to s, and finally (d) the socks which have already been paired. But altogether they occupy the same space which the original socks did.

However, the cost is that some socks will be lost. *How many?* It is easy to see first of all that if the socks come from at most d different shades, then at most d socks can be lost. Furthermore if the d shades are all non-matching then no socks can be lost at all.

To see what can happen here, suppose that he has socks of color dark gray, charcoal gray, black, dark blue, and midnight blue. Suppose a charcoal gray sock will match either a black sock or a dark gray sock. Other than that, no matches occur with other colors. Then at most 2 socks can be lost, regardless of the number n of socks. This is because at most one sock of each shade will remain unmatched, and neither a charcoal gray sock nor a blue sock can be one of these. For if a charcoal gray sock remained unmatched, there would also be another unmatched sock which was charcoal gray or dark gray or black. But a left over charcoal gray sock would match all three. Furthermore no blue socks will remain unmatched. So at most two socks, one black and one dark gray, could remain unmatched.

More generally, if d is the total number of shades, and d' is the size of the largest set of shades no two of which match (i.e. of the largest maximal anti-clique of the indistinguishability graph), then $d' \leq d$ and at most d' socks can remain unmatched. Also, if indistinguishability is an equivalence relation (as happens e.g. when $d \leq 2$) then no socks can remain unmatched and otherwise at most $d-1$. Finally, in such a case, with fixed d', the naive algorithm will run on the average in time linear in n, even though the worst case will be $O(n^2)$.

What if we do not have any bound other than $n/2$ on the number of shades? Even in this case we claim that at most a *third* of the socks can be lost. For suppose that we denote the original mate of a sock s by s^-, whence $s^{--} = s$. We also define $s^=$ to be the new mate[1] (if any) of sock s. The function $^-$ is total but $^=$ may, and usually will be, a partial function.

Now if sock s is thrown away, it means that s^- is (and has in fact already been) matched with some sock t $(= s^{-=})$. It could not have been thrown away, for s was available to be matched with it. Moreover, t^- must also have already been matched, otherwise *it* would have been available to be matched with s. So $t^=$ must exist. Proceeding this way we get a chain of socks $(s, s^-, t, t^- ...)$ ending with some *other*[2] unmatched sock, say u. The chain is obtained by alternately applying the functions $^-$ and $^=$. It is easy to see that if we had started this chain of reasoning with u instead of s, we would have got the same chain in reverse, ending up with s.

Let $X(s)$ be the set of (matched) socks between s and u, i.e. the socks $\{s^-, ..., u^-\}$. Then $X(s) = X(u)$. Moreover, $X(s)$ has an even number of elements and we just saw why it cannot be only two. Thus $X(s)$ has at least four

[1] Thus $s^{-=}$ is a little like the second husband of one's first wife.

[2] It is not hard to show that it cannot be s.

elements. Moreover, if v is some other thrown sock, then $X(v)$ is disjoint from $X(s)$. This is because the chains starting from s and v are both deterministic. If these chains overlap, they must both end up with u. In that case the chain beginning with u would travel backwards, ending up in both s and v, thereby forcing v to equal s. [3] Thus we see that for each sock s which is thrown away, there are at least four matched socks $X(s)$, and each set $X(s)$ is used only twice. This shows that the number of matched socks is at least twice the number of thrown socks, i.e. the number of thrown socks is at most 1/3 the total number of socks.

Our original example with six socks and three shades shows that this is the best possible in general for the naive algorithm and in fact we can create such a situation with an arbitrary number of shades. Suppose we have shades $\{d_0, ..., d_m\}$ such that the shade d_0 matches all shades and none of the other shades match each other. Also we have m socks of shade d_0 and two socks each of shade d_i for $i > 0$. Now if we happen to match the m socks of shade d_0, each with one sock of the shades d_i for $i > 0$, then when this is done, one sock each of shade d_i for $i > 0$ will be left over and nothing could be done with those. Thus to make sure that we match more than 2/3 of the socks will require a more sophisticated algorithm.

4 An $O(n^2)$ Algorithm Losing No Socks

: The greedy algorithm requires no extra space in the laundry room. Suppose Ravi does have space either in the laundry room or elsewhere, can he then do better? In particular can he pair up all his socks (unless the washing machine ate some)?

The following three algorithms, having progressively better running times, are all based on a straightforward yet crucial observation that a sock and its pair must "behave" identically. This assumes that a sock and its pair wash identically or indistinguishably. We capture this property in a a binary n-vector $\mathbf{v}_i$, associated with each sock s_i as follows: the jth entry of the vector is 1 if sock s_i is indistinguishable from sock s_j and 0 otherwise. The ith entry of $\mathbf{v}_i$ is always 1. The vectors for a sock and its pair must be identical. We simply need to detect identical vectors to obtain the class of socks with identical "behavior". This class must necessarily have an even number of socks and a sock from this class can be paired with any other sock from this class without creating a problem in the future.

It takes $O(n^2)$ time to compute all the vectors since every sock must be compared with every other sock. (The binary relation of indistinguishability, which must be specified to the algorithm somehow, could be given in the form of these vectors in the first place, in which case we would spend n^2 time merely reading the vectors in.)

Finding identical vectors is harder. A simple-minded approach would be

[3] It is possible of course that when the chains from s and v met, they were going in opposite directions. In that case a variant of our argument will give us $v = u$ rather than $v = s$. But v cannot be distinct from both s and u.

to examine all $n(n-1)/2$ pairs, spending time n to compare each pair. This method would then take $O(n^3)$ time.

If we plan ahead however, it might occur to us to first sort the vectors numerically, thinking of them as n-bit integers. Sorting n items requires $O(n \log n)$ comparisons, and each comparison of two n-bit vectors takes time n, for a cost of $O(n^2 \log n)$. Once sorted, we can find all identical pairs of vectors by examining the $n-1$ consecutive pairs at a cost of $O(n^2)$. So the preparatory sorting phase dominated the running time here: pairing up the socks went relatively quickly once they were sorted. The resulting $O(n^2 \log n)$ algorithm improves significantly on our simple-minded $O(n^3)$ method.

However we can do yet better here by taking advantage of the structure of this problem. The following method further reduces the time to $O(n^2)$.

Ravi constructs an $n \times n$ matrix M whose ith row is $\mathbf{v}_i$. He uses the following simple algorithm to partition the vectors where each partition corresponds to the class of identically "behaving" socks. Based on the first column of the matrix M, he partitions the rows (or socks) into two sets $S^k = \{\text{row } i | M[i,1] = k\}$, $k = 0, 1$. Next, using the second column, he partitions S^0 into S^{00}, S^{01} defined as follows $S^{0k} = \{\text{row } i | M[i,2] = k \text{ and } i \in S^0\}$, $k = 0, 1$. Similarly S^1 can be partitioned into sets S^{10}, S^{11}. He proceeds this way until all the columns have been considered. The maximum number of partitions is $n/2$ since a sock and its pair must "behave" identically. For any pair of rows i and j in the same partition, $\mathbf{v}_i = \mathbf{v}_j$ holds and the corresponding socks can be arbitrarily paired with any other element from the partition. This sorting or partitioning into identical vectors takes $O(n^2)$ time.

So, if Ravi does have some space to spare, he can have *all* his socks matched perfectly using the same order of time as the naive greedy algorithm.

The question still remains if there is a simple algorithm which matches *all* socks, or failing that, one which matches more than two thirds of the socks regardless of the number of shades? We leave the reader to ponder this issue. Meanwhile, we can only repeat Henry Ford's advice: *Buy socks of any color you like, as long as they are black!*

References

[1] K. Arrow, *Social Choice and Individual Values*, Wiley, (1963)

[2] M.A.E. Dummett, "Wang's paradox", *Synthese* **30** (1975), 301-304.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, 1979.

[4] J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," SIAM J. Comput. **2**, 1973, 225-231.

[5] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*, Series in Discrete Math and Optimization, Wiley Interscience, 1988.

[6] D. Knuth, "The Toilet Paper Problem", *American Math Monthly* **91:8**, Oct. 1984, 365-370.

[7] R. Parikh, "The Problem of Vague Predicates", *Logic, Language and Method*, Ed. Cohen and Wartofsky, Reidel (1982) 241-261.

[8] R. Parikh, "Vagueness and utility, the semantics of common nouns" *Linguistics and Philosophy* **17** 1994, 521-35.

[9] R. Parikh, "Language as social software", to appear in *Future Pasts: Perspectives on the Place of the Analytic Tradition in Twentieth Century Philosophy*, Ed. J. Floyd and S. Shieh, (1999)

[10] L. Wittgenstein, *Philosophical Investigations*, Basil Blackwell, (1953)