On Sorting Under Special Transpositions

Jici Huang
School of Computing
University of North Florida
Jacksonville, Florida 32224
Email: j.huang@unf.edu

Swapnoneel Roy
School of Computing
University of North Florida
Jacksonville, Florida 32224
Email: s.roy@unf.edu

***Abstract*—In this paper, we study a genome rearrangement primitive called *block moves*. This primitive as a special case of another well studied primitive *transposition*. We revisit the problem of BLOCK SORTING, which is a sorting problem under the primitive block moves in this work. BLOCK SORTING has been shown to be NP-Complete, and a couple of results have designed factor 2 approximation algorithms for the problem - the best known till date. However whether the problem is APX-Hard, or an improvement over the factor 2 approximation algorithms have been interesting open problems. We design a new factor 2 approximation algorithm for BLOCK SORTING. Our algorithm is equal to the best known in terms of approximation ratio, however, our approach is much simpler and is linear time ($O(n)$) as compared to the cubic ($O(n^3)$) and quadratic ($O(n^2)$) run-times of the existing algorithms for the problem.**

***Keywords*-Block Sorting; Transpositions; Optical Character Recognition;**

I. INTRODUCTION

Genome and other syntenic blocks rearrangements have become a topic of intensive study by phylogenists, comparative genomicists, and computational biologists: they are a feature of many cancers, must be taken into account to align highly divergent sequences, and constitute a phylogenetic marker of great interest. The mathematics of rearrangements is far more complex than for indels and mutations in sequences. Genome rearrangements have been modeled by a variety of primitives such as reversals, transpositions , block moves and block interchanges. The problem of genome rearrangement has been modeled as equivalent to sorting a permutation under a single or a combination of such primitives.

Genome is the entire DNA of a living organism. Gene is a segment of DNA that is involved e.g. in producing a protein, and its orientation depends on the DNA-strand that it lies on. Genome consists of chromosomes. Chromosomes are linear or circular.

Biologists are mostly interested in comparing species, for example:

1) In order to classify them
2) In order to explain evolution by reconstructing scenarios

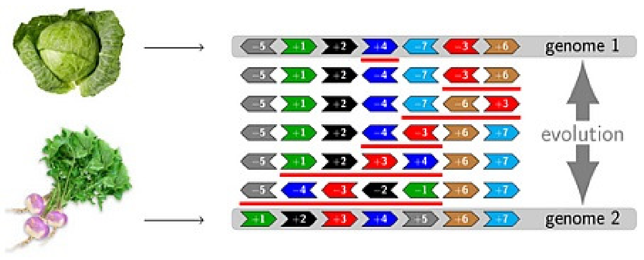

Figure 1. Genome Rearrangement

(Dis)similarity measures are needed to achieve these. The measures are usually based on the sequenced genomes.

The methodology used is to use a single or a combination of well defined primitives to transform one genome into the other. The number of primitive steps needed to transform one genome into another is a measure for the evolutionary distance between the two species.

GENOME REARRANGEMENT can be formally stated as:

GENOME REARRANGEMENT PROBLEM
INPUT: Genomes G_1, G_2, a set S of mutations.
GOAL: Find a shortest sequence of elements of S that transforms G_1 into G_2.

A related, simpler problem is to compute the *evolutionary distance* $d_S(G_1, G_2)$ (i.e. just the length of a shortest sequence). There are many variants of the above problems, depending on how genomes are modelled, and what (and how) mutations are taken into account, etc.

In the model we work genomes are represented by *permutations*.

Therefore in our model genomes:

1) Form ordered sequences of genes (or other segments), and
2) Only differ by order (no duplications or deletions).

The individual genes in the genomes can be numbered as we wish, so we assume the target genome is always the *identity* or sorted permutation $id = 12 \cdots n$. Hence GENOME REARRANGEMENT becomes a sorting problem where we wish to sort the other (starting) genome.

GENOME REARRANGEMENT (SORTING PERMUTATIONS)
INPUT: A permutation π, a well defined *sorting primitive* S.
GOAL: $d_S(\pi)$, a shortest sorting sequence
applying S on π at each step that transforms π into id.

Some well known primitives for genome rearrangements are transpositions [1], [2], [3], reversals (aka inversions) [4], [4], block moves (aka strip moves) [5], [6], [7], [8], block interchanges [9], prefix reversals [10], strip exchanges [11] etc. At times, a combination of more than one primitives is used to determine the distance (e.g. transreversals: a combination of transpositions and reversals) [12].

These sorting problems are *combinatorial optimization* problems in which the number of steps required to sort an arbitrary permutation is optimized. While the sorting by reversals [13], transpositions [2], prefix reversals [14], and block moves [6] have been proven to be NP-Hard, the computational complexity of the sorting by strip exchanges problem still remains open. The sorting by block interchanges problem is one of the very few such problems which has been proven to be polynomially solvable. An $O(n^2)$ exact algorithm exists for this problem [9].

Again, the similarity between two sequences will be measured by the minimum number of operations to transform one sequence into the other. Since the target sequence is always $1, 2, \cdots, n$ we view the problem as a sorting problem. But this is not a usual sorting problem which we are familiar with. Our sorting problem is to sort a sequence in such a way that the number of operations is *minimized*. In other words, we are interested in finding algorithms which always sort a sequence with minimum number of operations (transpositions, reversals, block moves, etc.).

We work on the primitive *block moves* in this paper. Block moves (aka strip moves) is a special case of another primitive *transpositions*. A block in a permutation π is a maximal sequence of consecutive elements that are also consecutive in the identity (sorted) permutation id. Sorting via block moves or BLOCK SORTING is to find the shortest sequence of blocks to be moved $bs(\pi)$ to sort a given permutation π. BLOCK SORTING is also a nontrivial variation of SORTING BY TRANSPOSITIONS. SORTING BY TRANSPOSITIONS arises in the context of genome rearrangements in computational biology. In transpositions, we are allowed to move any substring of π (not necessarily a block) to a different position at each step [1]. SORTING BY TRANSPOSITIONS is to compute the minimum number of such moves to sort π, it has been recently shown to be NP-Hard [2], the current best known algorithm has an approximation ratio of 1.375 [3], and is not known whether it admits a PTAS. It is not known yet whether BLOCK SORTING approximates SORTING BY TRANSPOSITIONS to any factor better than 3. However, it is known that optimal transpositions never need to *break* existing blocks [7]. This shows how the two problems are closely related. The study of the computational complexity of BLOCK SORTING therefore might provide us with more insight into the complexity of SORTING BY TRANSPOSITIONS.

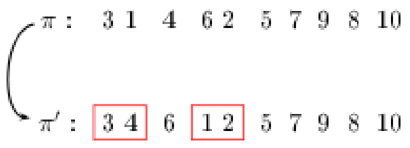

Figure 2. An example of a block move

The decision version of BLOCK SORTING has been shown to be NP-Hard [6]. After being known to be 3-approximable in [5], and [6], some results have designed 2-approximation algorithms [5], [7], [8]. However, no polynomial time approximation hardness results are known.

Our Contributions: Our main motivation is to improve the current best approximation ratio for BLOCK SORTING. We design another 2-approximation algorithm for the problem. Our algorithm has an approximation factor equal to the best known currently, but it runs in $O(n)$ (linear time) as compared to the $O(n^2)$ (quadratic time) of the algorithm of [5], and the $O(n^3)$ (cubic time) of the algorithm of [7]. Also our algorithm is much simpler to implement and analyze than the two existing algorithms.

The rest of the paper is structured in the following manner. Section. II introduces certain key concepts important to our analysis. Section III talks about our new approximation algorithm for BLOCK SORTING and analyzes it. Finally, Section IV summarizes the results and discusses future research directions.

II. PRELIMINARIES

The set $\{1, 2, \cdots, n\}$ is denoted by $[n]$, and let S_n denote the set of all permutations over $[n]$, and id_n the sorted or identity permutation of length n. The given permutation $\pi \in S_n$ to be sorted is represented as a string $\pi_1\pi_2\cdots\pi_n$ without loss of generality.

Block, Block Move. A block in a permutation π is a maximal sequence of consecutive elements that are also consecutive in the identity permutation id. In other word, a block is a maximal substring of a given permutation π, which is also a substring of the identity permutation id. As an example, the permutation 8 2 5 6 3 9 1 4 7 contains 8 blocks, and $\boxed{5\ 6}$ is the only block of length more than one. A *block move* picks up a block and places it elsewhere in the permutation. Fig. 2 shows a block move on a permutation π to obtain the permutation π'.

Block Sorting Schedule. A block sorting schedule $\mathcal{S}$ is a sequence of block moves such that performing the sequence of block moves on permutation π results in the identity permutation id. The length of a block sorting schedule is

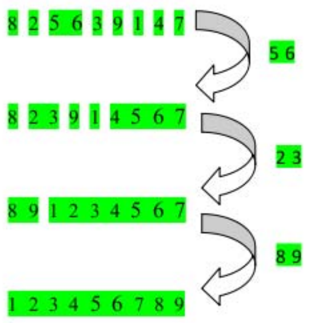

Figure 3. An example of a block sorting schedule

the number of block moves in the schedule. Block sorting distance $bs(\pi)$ is the number of a block moves in a minimum length block sorting schedule for π. A block sorting schedule is shown on permutation 8 2 5 6 3 9 1 4 7 in Fig. 3. The block moves are indicated at each step.

Kernel Permutation. Kernelizing a permutation is to replace each block by an element which is its rank among the blocks. Hence the permutation 7 2 5 3 8 1 4 6 is equivalent to 8 2 5 6 3 9 1 4 7. This can be done because we do not *break* blocks once they get joined to form larger blocks in a block move. The permutation where each block of π is replaced by its rank among the blocks is termed as a kernel permutation $ker(\pi)$ [8] for π (aka a *reduced* permutation in [6]). In $ker(\pi)$, each block is of length 1.

BLOCK SORTING can be formally stated as:

> BLOCK SORTING PROBLEM
> INPUT: A permutation π and an integer m.
> QUESTION: Is $bs(\pi) \leq m$?

In [8], it has been formally proved that block sorting π is equivalent to block sorting its kernel $ker(\pi)$. That is $bs(\pi) = bs(ker(\pi))$. Also it was shown in [8], that in an optimal block sorting sequence, we never need to break apart an existing block at any step. That is the block sorting distance remains the same, even if we allow block sorting moves which do not necessarily join blocks, or which break any previously joined blocks. Hence we treat blocks as *single elements* in this work without loss of generality, and use the relations $<$, and $>$ on blocks in their usual senses. For a block $a \in \pi$, we denote its position in π by $\pi(a)$.

Reversal. In a permutation π, a pair of consecutive elements (blocks) a and b form a reversal if $a > b$, and $\pi(b) = \pi(a) + 1$. Denote the number of reversals in π by $rev(\pi)$. In [8], it has been shown that a block sorting sequence of length $rev(\pi)$ is optimal, since the block sorting distance $bs(\pi) \geq rev(\pi)$. As an example, the reversals in the permutation 8 2 5 6 3 9 1 4 7 are $\{(8, 2), (\boxed{5\ 6}, 3), (9, 1)\}$.

Inversion. In a permutation π, a pair of elements (blocks) a and b form an inversion if $a = b + 1$, and $\pi(a) > \pi(b)$. Inversions are also called *dual reversals* in [8]. Denote the number of inversions in π by $inv(\pi)$. In [8], it has been shown that a block sorting sequence of length $inv(\pi)$ is optimal, since the block sorting distance $bs(\pi) \geq inv(\pi)$. As an example, the inversions in the permutation 8 2 5 6 3 9 1 4 7 are $\{(1, 2), (4, \boxed{5\ 6}), (7, 8)\}$.

Run. Another important concept in a permutation π is a *run*. The blocks $\{a, a+1, a+2, \ldots, a+r\}$ form a run of length r if, r is the largest values such that $\pi(a) < \pi(a+1) < \pi(a+2) < \ldots < \pi(a+r)$. Then number of runs in π is denoted by $runs(\pi)$. As an example, the runs in the permutation 8 2 5 6 3 9 1 4 7 are $\{(1), (2, 3, 4), (\boxed{5\ 6}, 7), (8, 9)\}$.

III. A 2 APPROXIMATION ALGORITHM FOR BLOCK SORTING

Lower Bounds. For permutation π, let

1) n be the number of blocks in π,
2) i be the number of inversions in π, i.e. $inv(\pi) = i$,
3) b be the number of reversals in π, i.e. $rev(\pi) = b$,
4) r be the number of elements (blocks) in the *longest* run of π.

Then $bs(\pi) \geq max\{i, b, n-1-(i+b)\}$ [6]. Algorithm 1 describes our new approximation. In simple words, we find the longest run in π, we keep the r elements of that run *intact*. We move every other block to obtain the sorted permutation id.

```
Data: The input permutation $\pi$
Result: The sorted permutation $id$
Find the longest run $R$ in $\pi$;
while $\pi$ is not sorted do
    /* For a block $B \notin R$                */
    if $B$ is not in proper position then
        Move $B$ either before $B+1$ or after $B-1$;
    end
end
```

Algorithm 1: Approximate block sorting.

Theorem 1. *Algorithm 1 is a 2 approximation algorithm for* BLOCK SORTING.

Proof: The above algorithm takes at most $n - r$ moves where n is the number of blocks in π. The approximation ratio is thus $\frac{n-r}{max\{i,b,n-1-(i+b)\}}$. We show this ratio to be at most 2 by exploring different values of $n - 1 - (i + b)$.

1) **Case I**. $n - 1 - (i + b) < 0$. In this case we have $n - 1 < (i + b)$. Since $n - r \leq n - 1$, therefore $\frac{n-r}{max\{i,b,n-1-(i+b)\}} < \frac{i+b}{max\{i,b,n-1-(i+b)\}} \leq 2$ depending on whether $max\{i, b, n - 1 - (i + b)\}$ equals i or b. We note both i and b are $\geq n - 1 - (i + b)$ in this case, since both are always ≥ 0.

2) **Case II**. $n - 1 - (i + b) \geq 0$. Let us consider the following sub-cases.

   a) **Case IIa**. $n - 1 - (i + b) < i + b$. Therefore $n - 1 < 2(i + b)$. Let us assume $\frac{n-r}{n-1-(i+b)} \leq 2$. Then we have $n - r \leq 2(n - 1) - 2(i + b)$. Substituting $2(i + b)$ by $n - 1$ we have $n - r < 2(n - 1) - (n - 1)$. This implies $n - r < n - 1$ must be true for the condition $\frac{n-r}{n-1-(i+b)} \leq 2$ to be true. We note $n - r < n - 1$ is true for $r > 1$. In general we have $r \geq 1$ (since r is the number of blocks in the longest run of π). We further note $r = 1$ would imply π to be either sorted (equal to id) or be the *reverse* permutation (falls under Case I).

   b) **Case IIb**. $n - 1 - (i + b) \geq i + b$. Therefore $n - 1 \geq 2(i + b)$, or $\frac{n-1}{2} \geq (i + b)$. Here we have $\frac{n-r}{n-1-(i+b)} \leq \frac{n-1}{n-1-(i+b)} \leq \frac{n-1}{n-1-\frac{n-1}{2}} = 2$. ■

Theorem 2. *Algorithm 1 runs in linear time $O(n)$, where n is the number of blocks in π.*

Proof: Finding the longest run in π takes $O(n)$ time. The remaining part of the algorithm takes at most $n-r$ block moves which is again $O(n)$. Therefore the total runtime is $O(n)$. ■

IV. Conclusion

In this paper, our key contribution is a simpler 2 approximation algorithm for the Block Sorting problem. This might lead to an algorithm with a better approximation factor, something which has been open for over a decade. Also it would be interesting to know whether Block Sorting admits a PTAS or is APX-Hard. The latter result might lead to similar results for the more generalized Sorting by Transpositions problem.

Acknowledgment

The authors thank the anonymous reviewers of IEEE BIBE 2014 to help improve the quality of this paper.

References

[1] V. Bafna, Pavel, and A. Pevzner, "Sorting by transpositions," *SIAM Journal on Discrete Mathematics*, 1998.

[2] L. Bulteau, G. Fertin, and I. Rusu, "Sorting by transpositions is difficult," in *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I*, ser. ICALP'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 654–665. [Online]. Available: http://dl.acm.org/citation.cfm?id=2027127.2027197

[3] I. Elias and T. Hartman, "A 1.375-approximation algorithm for sorting by transpositions," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 3, no. 4, pp. 369–379, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1109/TCBB.2006.44

[4] V. Bafna and P. A. Pevzner, "Genome rearrangements and sorting by reversals," *SIAM J. Comput.*, vol. 25, no. 2, pp. 272–289, Feb. 1996. [Online]. Available: http://dx.doi.org/10.1137/S0097539793250627

[5] W. W. Bein, L. L. Larmore, L. Morales, and I. H. Sudborough, "A faster and simpler 2-approximation algorithm for block sorting," in *Proceedings of the 15th International Conference on Fundamentals of Computation Theory*, ser. FCT'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 115–124. [Online]. Available: http://dx.doi.org/10.1007/11537311_11

[6] W. Bein, L. Larmore, S. Latifi, and I. Sudborough, "Block sorting is hard," in *Parallel Architectures, Algorithms and Networks, 2002. I-SPAN '02. Proceedings. International Symposium on*, 2002, pp. 307–312.

[7] M. Mahajan, R. Rama, and S. Vijayakumar, "Block sorting: A characterization and some heuristics," *Nordic J. of Computing*, vol. 14, no. 1, pp. 126–150, Jan. 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1515784.1515790

[8] M. Mahajan, R. Rama, V. Raman, and S. Vijaykumar, "Approximate block sorting," *International Journal of Foundations of Computer Science*, vol. 17, no. 02, pp. 337–355, 2006. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0129054106003863

[9] D. A. Christie, "Genome rearrangement problems," Ph.D. dissertation, 1999.

[10] W. H. Gates, "Bounds for sorting by prefix reversal," *Discrete Mathematics*, pp. 47–57, 1979.

[11] S. Roy and A. Thakur, "Towards construction of optimal strip-exchanging moves," in *Bioinformatics and Bioengineering, 2007. BIBE 2007. Proceedings of the 7th IEEE International Conference on*, Oct 2007, pp. 821–827.

[12] D. Christie and R. Irving, "Sorting strings by reversals and by transpositions," *SIAM Journal on Discrete Mathematics*, vol. 14, no. 2, pp. 193–206, 2001. [Online]. Available: http://dx.doi.org/10.1137/S0895480197331995

[13] A. Caprara, "Sorting by reversals is difficult," in *Proceedings of the First Annual International Conference on Computational Molecular Biology*, ser. RECOMB '97. New York, NY, USA: ACM, 1997, pp. 75–83. [Online]. Available: http://doi.acm.org/10.1145/267521.267531

[14] L. Bulteau, G. Fertin, and I. Rusu, "Pancake flipping is hard," in *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, ser. MFCS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 247–258. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32589-2_24