Emerge-Sort: Swarm Intelligence Sorting

Dimitris Kalles[1,2], Vassiliki Mperoukli[1], and Andreas Papandreadis[2]

[1] Hellenic Open University, Patras, Greece
kalles@eap.gr, vickybergr@yahoo.gr, bp@hol.gr
[2] Open University of Cyprus, Nicosia, Cyprus

Abstract. We examine sorting on the assumption we do **not** know in advance which way to sort. We use simple local comparison and swap operators and demonstrate that their repeated application ends up in sorted sequences. These are the basic elements of Emerge-Sort, an approach to self-organizing sorting, which we experimentally validate and observe a run-time behavior of $O(n^2)$.

1 Introduction

Sorting has been one of the first areas of computer science to showcase efficient algorithms to stand the test of time. Our key motivation to examine sorting is to investigate what is the minimum structure of local operators whose repetitive application ends up in sorted sequences. We relax the assumption that we know which way to sort, yet arrive at a sorted sequence by simple local interactions, which are also able to address on-the-fly modifications of the sequence-to-be-sorted. Accordingly, our contribution lies with the development of several such operators and the validation of their **emergent** sorting efficiency independently of any global sorting direction bias.

A simple example may help to visualize such a context. Consider a database which is distributed across several sites, with each site maintaining a sorting that best suits its local needs (either largest-first or smallest-first). If there is a need to redistribute objects across the sites, according to their order, so as to avoid asking all sites where an object might be located, one can either opt for some sort of centralized control over who-comes-first or, simply, let each object re-arrange itself in the local neighborhood. Additionally, when new objects are inserted dynamically into any of the available sites, one can either explicitly direct each one of them to the correct database partition or, simply, wait until the object finds its way to a partition. Similar problems have been long studied in the distributed computing literature [1][2][3], have attracted the attention of the multiagent systems community [4][5] and the consensus reaching community [6].

Locality is central to the swarm intelligence and game theory areas too. Swarm intelligence has gathered significant momentum since many traditional problems (graph partitioning and clustering, among others) have been recast in terms of swarm behavior. Swarm intelligence is at the junction of randomized behavior and local operations and has been demonstrated to solve difficult problems such as task scheduling [7]. Now, emergent sorting is a behavior that has been documented in insect societies and modeled via swarm intelligence principles [5][8], albeit in an unconventional way; therein, sorting takes place in 2-D and consists of forming concentric circles where items of similar size are at roughly the same distance from the centre. 2-D sorting has received relatively scant attention since it is usually seen as a (difficult) case of clustering, another classic problem that has been also recently addressed with swarm intelligence [9]. It is interesting to note that swarm intelligence, beyond emergent sorting [4], has been also related to autonomic computing research, mainly through the observation of autonomic computing principles in biology inspired systems and the transfer of relevant concepts to problems in distributed computing [10].

Game theory, on another front, directs huge research effort at studying various computational games from the point of view of reaching a Nash equilibrium; therein, one usually favors algorithms that make little use of global knowledge and where agents act competitively yet manage to converge to a state that satisfies them all [11]. In such settings, one asks what expense such anarchy incurs when seen from an optimization point of view; in other words, if one could centrally design what each agent will do, it is interesting to know how much effort one would save. The beauty of localized behavior in that setting usually comes from the appreciation of its robustness and graceful behavior in adversarial contexts as well. The similarity to our problems is quite straightforward: we are, what is the cost of not knowing which way we need to sort and how can we induce our sequences to self-sort.

Seen from a more conventional computer science perspective, swarm intelligence draws on several aspects of distributed computing [1][12][13], where, however, the underlying algorithm usually assumes a sequence of distinct passes over the data to achieve sorting. It is interesting to see that $O(n^2)$ seems to be the minimum one has to pay for such sorting but we note that our local operators achieve such performance without any knowledge of further steps. From that perspective, we view cellular automata to be also related to our approach [14][15, also taking into account the genetic discovery of interesting behaviors for such automata [16][17] and their influence on distributed co-ordination in consensus reaching problems [18][19].

The rest of this paper is structured in three sections. The next one is the core of the paper: we sketch out the basic principles that have lead to Emerge-Sort by showing how various modifications in simple local operators influence their capacity to finally deliver sorted sequences. Following that, we show a brief experimental validation of these fundamental concepts and, finally, we conclude the paper by drawing key connecting lines from our research to swarm intelligence and by setting forth the key future research viewpoint.

2 Developing Emerge-Sort

There are six possible ways to permute a triple (a,b,c) of distinct numbers (Fig. 1); additionally, un-sorted triples require a single swap between two elements to turn into sorted ones.

It turns out that this simple operator cannot scale to arbitrarily selected triples of any unordered sequence (it is not difficult to devise a counter-example demonstrating eternal oscillation between two sequences), so the next best attempt is to minimally extend this operator.

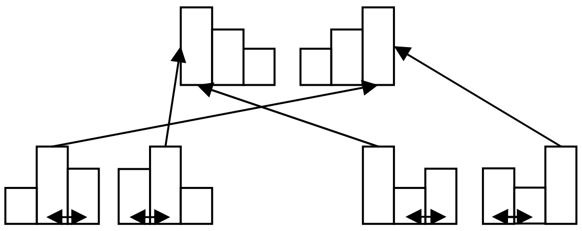

Fig. 1. Possible permutations and swaps to sort any three numbers

Interestingly, the extension is quite simple: sort the triple in the direction indicated by a majority vote among all triple members' direction preferences. Such direction preference (bias) can be initialized at random for each member. In effect, the new local sorting operator (which, we stress, is confined to the triple's limits):

- First, establishes a preference for a sorting direction,
- Then, enforces that direction by locally sorting the triple,
- Finally, modifies the sorting direction bias of the minority vote items.

So, we have substituted the "to make as few moves as possible" approach with a still very simple one, namely "to make as few direction bias changes as possible" (which, of course, means that the arrows shown in Fig. 1, about how permutations are turned into sorted triples, are no longer valid). The new approach creates overlapping triples that compete for setting the sorting direction of the sequence. But, this suffices for the sequence to be eventually sorted.

2.1 A Proof of Concept

As a convention, let us use *A* to denote an ascending sorting bias (for left-to-right), and *D* to denote a descending sorting bias (again, for left-to-right). Let us also assume that we have a sequence of length *n* to sort. A triple is uniquely identified by the location of its middle element, *i*, with $1 \leq i \leq n$ and we employ a simple wrapping that arranges the sequence in a circle (ring).

Now, assume that at initialization, all numbers in the sequence have equal probability of having a sorting bias direction of *A* or *D*. Note that the sorting bias in a triple only conveys information as to what sorting direction it *plans* to employ.

We now show an example of a neck-to-neck competition for establishing the bias; it will then be easier to develop a general probabilistic argument.

Let us assume that a sequence has the following bias distribution: *DDDDAAAA*. Some locations (underlined) belong to triples with competing biases: $\underline{\boldsymbol{D}}DD\underline{\boldsymbol{D}}\boxed{\underline{\boldsymbol{A}}}AA\underline{\boldsymbol{A}}$; let us examine one such contest point at the centre of triple 5. For this point, triples 4, 5 and 6 compete to set its bias. Even if all triples act purely parallel, the one to fire last gets the final opportunity to modify that location's bias. If we examine all contest points for the particular configuration set out above and enumerate all possibilities of alternative triple firing orders, it turns out that, there is a 1/3 probability that there will again be four *A*s and four *D*s and there is a 2/3 probability that either *A*s or *D*s will overwhelm the other.

However, establishing a majority of one sorting direction bias will, quite probably, create more triples of the same bias, increasing the probability that all triples will end up with the same bias. Eventually, this sorts the sequence.

2.2 A Global Sorting Direction Bias Delivers a Sorted Sequence

When all triples have the same sorting direction bias, any triple that is being picked to sort itself will contribute to the overall sorting.

A simple measure of unsorted-ness of any sequence is the number of alternating runs, as defined by the times one has to change the upwards or downwards direction when traversing the sequence, end-to-end. For example, the sequence (3, 5, 6, 2) has 2 alternating runs; a triple can have at most 2 alternating runs.

Assume that one has achieved a global sorting direction bias of A (of course, no triple is aware that its bias is also a global one). Then, any time a triple is locally sorted, it either decreases the number of alternating runs of its extended neighborhood (and of the whole sequence, as a result) or it leaves that number unchanged. Each alternative has a probability of ½.

Now, consider a sequence of length n, where, in one parallel round, all triples sort themselves. Of course, we have no guarantee that the actual number distributions in these triples will mean that a particular local sorting will decrease the overall number of alternating runs with a probability of ½. However, we can approximate as $1/2^n$ the probability that no individual triple sorting will affect the overall sequence's number of alternating runs. Since this number converges very fast down to 0, an incremental improvement (decrease) of 1 in the unsorted-ness measure is almost certain to happen.

Note that, whenever large chunks of the sequence are already sorted, there exist fewer points which serve as ends of the alternating runs and the previous argument does not hold. But, to compensate for that, we note that the actual numbers of alternating runs is now much smaller itself, compared to when the sequence is shuffled. The number of alternating runs can be viewed as a potential function, which, at every point, strictly does not increase.

With a settled sorting direction bias, the maximum distance one number needs to travel is n-1, so $O(n)$ parallel steps suffice to sort the sequence. If we allow each triple to fire independently of each other and randomly, then it may take $O(n^2)$ individual triple local sorting operations to deliver the final sorted sequence.

2.3 Converging to a Global Sorting Direction Bias

Considering any triple in isolation, just one operation suffices to switch it to a common bias; any minority item cannot avoid having its flag overturned.

The next more complex case is to consider a quadruple with evenly distributed flags. When examined in triples, it will also have a global direction bias settled fast; the first step will change one flag and, then, a single flag cannot survive.

Things become more complicated with longer sequences. When examining an *ADDAA* quintuple, oscillations can occur *ad nauseam* between alternating configurations (for example, if we first examine a triple centered around a *D* item, then the majority of flags features a *D*, which can change again if we examine a triple centered around one of the remaining *A*s).

The probability of oscillations strictly decreases the longer we examine local triples and eventually one ends up in a sequence with a common flag; this happens because we have a Markov chain with two equilibrium distributions, all *A*s or all *D*s, whose stationary distribution (denoted as M^i for a square *n*x*n* matrix, with *n* being the length of sequence) converges to a single column of 1's. This is a probabilistic guarantee that all sub-sequences with the same flag will be eventually washed out.

$$M^{i}_{nxn} = \begin{bmatrix} 1 & 0 & & \dots & & 0 \\ 1 & 0 & & \dots & & 0 \\ \underset{i\to\infty}{1} & \underset{i\to\infty}{0} & \underset{i\to\infty}{0} & \dots & \underset{i\to\infty}{0} & \underset{i\to\infty}{0} \\ \vdots & \underset{i\to\infty}{0} & \ddots & \dots & \iddots & \underset{i\to\infty}{0} \\ \vdots & \vdots & & \vdots & & \vdots \\ \underset{i\to\infty}{1} & \underset{i\to\infty}{0} & \underset{i\to\infty}{0} & \dots & \dots. & \underset{i\to\infty}{0} \end{bmatrix}$$

Brief experimentation with several variants of the matrix above revealed that they converged within 1% of the stationary form in $O(n^3 \log n)$. But, as we shall see in the experimental section below, this is an unwarranted pessimistic estimate when compared to the average sorting time for a variety of sequences.

3 Experimental Validation

To validate the principle of Emerge-Sort we describe a series of experiments we have carried out across a range of input sizes and with several initial distributions of *A* and *D* biases.

We review in Table 1 the results for $n = 128$. Each line corresponds to 100 experiments. The ***% A*** column indicates the percentage of elements that have been initialized with an *A* sorting direction bias. It is reasonable, of course, that when a large majority of elements has the same sorting direction bias, there is a high probability that the sequence will eventually be sorted according to that bias. The ***% sorted Ascending*** column confirms just that.

Note that, as earlier argued, it is also reasonable that the faster all sequence elements converge to a common sorting direction bias, the faster it will be able to eventually sort itself. This is confirmed by the ***# Rounds*** column, which reports the individual number of triples examined. It is also confirmed by the **# Moves** columns, which shows how many numbers are moved per round on average. Since triples are fired at random, rounds are measured in terms of individual triple inspections.

Table 1. Results for n = 128 (100 experiments)

% A	% sorted Ascending	# Rounds	# Moves
99	100	4614	51
95	100	5163	52
90	99	6099	53
80	97	7778	58
70	96	9346	64
60	68	10697	74
55	63	10985	76
50	54	10949	76

We present in Fig. 2 a clear $O(n^2)$ pattern for larger values of n.

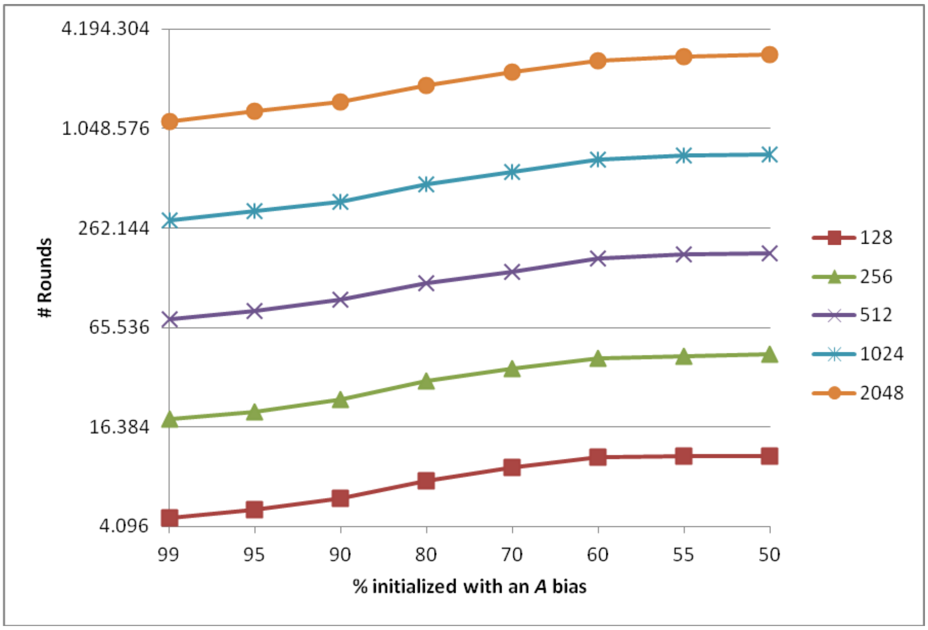

Fig. 2. Results for n = 128, 256, 512, 1024, 2048

4 Conclusions and Future Directions

We have described the concepts and a brief experimental validation of Emerge-sort, a sorting algorithm that does not depend on being told which way a sequence should be sorted and yet manages to sort that sequence, usually alongside a dominant preference among the sequence numbers, based on randomly applied simple local operators.

Emerge-sort closely follows the concepts of ant based algorithms, where sequences of numbers are societies of individuals who act locally and are unaware of the consequences of their actions on the sequence as a whole. Therein no shared knowledge, control or logic exists; individuals are only triggered into action via messaging by objects within their reach. Direction bias flags act as pheromones that are laid on the ground (this is the *stigmergy* concept), inviting others to follow a certain direction (this is the *chemotaxis* concept). Although the society has no hierarchies, large numbers can be viewed as foraging agents who influence others by helping settle direction issues (much like how taller children act when a group of children self-arranges itself in a circle according to height in a schoolyard). Furthermore, when a triple finds itself ordered it remains idle, being oblivious to a grander common global goal. But this delivers the *n*/log*n* penalty which this society pays for not having an *a priori* leader to settle the direction issue.

Research in Emerge-sort started as an experimental effort into investigating the dynamics of local operators regarding their ability to generate order. But, in computing, the trade-off between local operators and co-ordination has been studied in several directions, ranging from distributed systems to approximation algorithms and to algorithmic game theory, also drawing on cellular automata from computational physics. So, we expect that by studying Emerge-Sort alongside established and solidly founded paradigms we will be able to research termination and complexity issues of its algorithmic nature using arguments that may have been applied in these domains; this will also allow us to gain more insight into the crossing lines between such paradigms.

Acknowledgements. Loizos Michael and Ioannis Stamatiu have offered valuable comments on mathematical properties of card shuffling and ring networks. Emerge-Sort is fully implemented in Java and is available on demand for academic purposes. An earlier version was implemented in Maple. It is temporarily hosted at http://student-support2.ouc.ac.cy:8080/emergesort/index.html, where a few variants are also demonstrated.

References

1. Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N.: Sorting and election in anonymous asynchronous rings. Journal of Parallel and Distributed Computing 64, 254–265 (2004)
2. Prasath, R.: Algorithms for Distributed Sorting and Prefix Computation in Static Ad Hoc Mobile Networks. In: 2010 International Conference on Electronics and Information Engineering, vol. 2, pp. 144–148 (2010)
3. Israeli, A., Jalfon, M.: Uniform Self-Stabilizing Ring Orientation. Information and Computation 104(2-3), 175–196 (1993)
4. Casadei, M., Gardelli, L., Viroli, M.: Collective Sorting Tuple Spaces. In: 11th International Workshop on Cooperative Information Agents, Delft, The Netherlands, pp. 255–269 (2006)
5. Casadei, M., Menezes, R., Viroli, M., Tolksdorf, R.: Using Ant's Brood Sorting to Increase Fault Tolerance in Linda's Tuple Distribution Mechanism. In: Klusch, M., Hindriks, K.V., Papazoglou, M.P., Sterling, L. (eds.) CIA 2007. LNCS (LNAI), vol. 4676, pp. 255–269. Springer, Heidelberg (2007)

6. Bénézit, F., Thiran, P., Vetterli, M.: The Distributed Multiple Voting Problem. IEEE Journal of Selected Topics in Signal Processing 5(4), 791–804 (2011)
7. Bonabeau, E., Theraulaz, G., Deneubourg, J.-L.: Fixed Response Thresholds and the Regulation of Division of Labour in Insect Societies. Bulletin of Mathematical Biology 60, 753–807 (1998)
8. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
9. Handl, J., Meyer, B.: Ant-based and swarm-based clustering. Swarm Intelligence 1(2), 95–113 (2008)
10. Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G., Ducatelle, F., Gambardella, L., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology to distributed computing. ACM Transactions on Autonomous and Adaptive Systems 1(1), 26–66 (2006)
11. Koutsoupias, E., Papadimitriou, C.: Worst case equilibria. In: Annual Symposium on Theoretical Aspects of Computer, pp. 404–413. Springer (1999)
12. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM 17(11), 643–644 (1974)
13. Loui, M.C.: The complexity of sorting on distributed systems. Information and Control 60, 70–85 (1984)
14. Gonzaga de Sa, P., Maes, C.: The Gacs-Kurdyumov-Levin automaton revisited. Journal of Statistical Physics 67(3-4), 507–522 (1992)
15. Gordillo, J.L., Luna, J.V.: Parallel sort on a linear array of cellular automata. In: International Conference on Systems, Man and Cybernetics, vol. 2, pp. 1903–1907 (1994)
16. Mitchell, M., Hraber, P.T., Crutchfield, J.P.: Revisiting the edge of chaos: Evolving cellular automata to perform computations. Complex Systems 7, 89–130 (1993)
17. Andre, D., Bennett III, F.H., Koza, J.R.: Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: First Annual Conference on Genetic Programming, Stanford, CA, pp. 3–11 (1996)
18. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Gossip algorithms: Design, analysis and applications. In: 24th Annual Joint Conference of the IEEE and Communication Societies, pp. 1653–1664 (2005)
19. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: IEEE Conference on Foundations of Computer Science, pp. 482–491 (2003)