A Mixed Evolutionary-Statistical Analysis of an Algorithm's Complexity

Carlos Cotta[1]*, Pablo Moscato[2]

[1] Dept. Computer Science, ETSI Informática (3.2.49), Universidad de Málaga, Campus de Teatinos, 29071-Málaga, Spain

[2] Dept. de Engenharia de Computação e Automação Industrial, Universidade Estadual de Campinas, C.P. 6101, Campinas, SP, CEP 13083-970, Brazil

[1]ccottap@lcc.uma.es [2]moscato@densis.fee.unicamp.br

Abstract: A combination of evolutionary algorithms and statistical techniques is used to analyze the worst-case computational complexity of two sorting algorithms. It is shown that excellent bounds for these algorithms can be obtained using this approach; this fact raises interesting prospects for applying the approach to other problems and algorithms. Several guidelines for extending this work are included.

Keywords: Algorithms, Statistical Analysis, Computational Complexity, Evolutionary Computing

1 Introduction

The central question we would like to address can be stated as: "Is it possible to systematically develop computer-assisted, adversary-based approaches aimed to help with the analysis of algorithms and their concrete implementations in actual code?" All practitioners and researchers know that the worst-case asymptotic analysis of algorithms is one of the most successful paradigms in computing. However, the process of finding the worst-case instance for the particular algorithm under scrutiny is a rather complex process that highly depends on inspiration. Needless to say, it is a hard task that also involves mathematical skills and good working knowledge of the many theoretical tools that have been developed during the past century.

We will illustrate the use of evolutionary algorithms [1] to assist the task of finding worst-case instances. Evolutionary algorithms are heuristic search techniques based on the principles of natural evolution, namely adaptation and survival of the fittest. Starting from a pool of random solutions for the problem at hand, an iterative process is performed comprising selection (propagation of the best solutions in the pool according to an *ad-hoc* user-designed "fitness" function), reproduction (generation of new trial solutions by combining and modifying propagated solutions) and replacement (substitution

*Carlos Cotta is the author to whom correspondence should be addressed.

of the worst solutions in the pool by the newly created ones). When a termination criterion is met (usually reaching a fixed number of solution evaluations), the best solution found is returned.

We propose the use of this type of heuristic search processes to find hard instances for particular algorithms. By varying the size of the instances sought, we can obtain a data set comprising pairs $(size_i, complexity_i)$. Subsequently, we can apply statistical tools in order to find a functional expression $complexity_i = f(size_i)$. This latter part of our approach is thus related to Chakraborty and Choudhury's work [2]. Actually, this work extends this cited approach by (a) tackling the worst-case analysis, rather than the analysis of random instances, and (b) considering the case in which the shape of the functional expression f is not completely known.

As in [2], we selected sorting as the algorithmic benchmark. The motivation is twofold: firstly, it is a problem studied in depth from both a theoretical and experimental point of view; secondly, the analysis of the most conspicuous algorithms for its solution still deliver us a non-trivial test suite. In essence, in this article we are responding in a systematic way to challenges such as that proposed by Sedgewick in [6] (page 110) that regarding an $O(n^{3/2})$ algorithm for ShellSort said:

> *"The proof of this property is beyond the scope of this book, but the reader may not only appreciate its difficulty but also be convinced that shellsort will run well in practice by attempting to construct a file for which shellsort runs slowly."*

The comparison of the elaborate mathematical analysis required to obtain complexity bounds for this algorithm with the relative simplicity of the presented approach will shed some light on the potential usefulness of this method.

2 Statistical Analysis on Experimental Results

The sorting algorithms considered in this work are *BubbleSort*, and *ShellSort*. A problem instance for a sorting algorithm is a sequence of elements to be sorted according to a key-value. Without loss of generality, this sequence is chosen as a permutation of the set of natural numbers $\{1, 2, \cdots, n\}$, where n is the size of the instance. Thus, the task of the evolutionary algorithm will be to find hard permutations for the particular sorting algorithm being analyzed.

We have considered a standard steady-state genetic algorithm (GA) ($PoolSize = 100$, $p_c = 0.9$, $p_m = 1/n$, $maxevals = 100n^2$) using ranking selection ($\eta = 2.0$), the *uniform cycle crossover* operator (UCX) [4] for recombination, and the *swap* operator [5] for mutation. We purposefully decided to use a very simple evolutionary scheme to test the feasibility of the original idea. We refer to [3] for the implementation details of a GA. Our fitness function (i.e. the objective function to be maximized) is the number of elemental operations (element exchanges in this case) the sorting algorithm makes for finding the ordered sequence given an initial permutation.

The BubbleSort algorithm is based on making $O(n)$ passes through the list, performing pairwise comparisons (and exchanges if appropriate) between elements in consecutive positions. This is a very simple, standard and well-known algorithm, so we refer to [2] or any standard textbook for the actual code of the algorithm. It can be easily argued that this algorithm has a quadratic worst-case behavior. Furthermore, the exact worst-case complexity is known to require $n(n-1)/2$ exchanges.

The GA has been applied to this algorithm for instance sizes ranging from 10 up to 100 elements. A total number of 20 runs have been done on each size, and the hardest instance of these 20 runs has been kept. Subsequently, a least-squares fit to a 2$^{\text{nd}}$-degree polynomial is sought. Fig. 1 (left) shows the results: there exists an exact match between the experimental data provided by the worst-case instances generated by the GA (for different values of n, the X-axis) and the parabola $y = an^2 + bn$ for $a = 0.5$ and $b = -0.5$. Trying higher-order polynomials yields the same result (actually, the residuals are zero since empirical data are located right on the mentioned curve). In essence, the EA has matched the exact asymptotic worst-case behavior for BubbleSort, consistently finding the precise hardest-instance (the permutation $\langle n, n-1, n-2, \cdots, 3, 2, 1\rangle$).

Fig. 1 here

ShellSort [7] was one of the earliest sorting methods to be discovered. Based on insertion sort, this algorithm proceeds left to right through the list in a sequence of interleaved passes. These passes are done on the basis of a certain increment sequence. The `C` code for ShellSort is the following:

```
void ShellSort (int list[], int length) {
     int incs[14] = { 2391484, 797161, 265720, 88573,
                      29524, 9841, 3280, 1093, 364,
                      121, 40, 13, 4, 1 };
     for (int k = 0; k < 14; k++)
         for (int h = incs[k], i=h; i<length; i++) {
             int v=list[i];
             int j=i;
             while ((j>=h) && (list[j-h]>v)) {
                   list[j] = list[j-h];
                   j -= h;
             }
             list[j] = v;
         }
}
```

Despite its apparent simplicity, this algorithm has deserved an enormous amount of theoretical and empirical studies. In this sense, one of the key points is the study of different increment sequences and the determination of the corresponding complexity bounds. The sequence we have considered here ($h_0 = 1$, $h_{i+1} = 3h_i + 1$) was proposed by D. Knuth, and it is frequently used since it is easy to compute, uses relatively few (about $\log_3 n$) increments, and does well in empirical studies.

The results of the GA applied to ShellSort for sizes ranging from 10 up to 100 are shown in Fig. 1 (right). As it can be seen, the experimental data can be fit both to a parabola $y = an^2 + bn$ ($a = 0.0902$, $b = 8.5189$) and to a power function $y = an^b$ ($a = 2.0812$, $b = 1.4670$). Higher-order polynomials have been tested as well, yielding unrealistic results: the highest-order coefficient is negative for 3rd-degree, 4th-degree, and 5th-degree polynomials. Clearly, these results are due to overfitting, and cannot represent the known monotonic growth in the computational cost for increasing list sizes.

Notice that despite the fit to the power function being a very accurate estimate for the optimal $b = 3/2$, the model measures are fairly the same for both the parabola and the power function (the standard errors are $s_e = 33.4373$ and $s_e = 33.7594$ respectively, being the coefficients of determination $R^2 = 0.99627$ and $R^2 = 0.99620$ respectively), so it is difficult to ascertain which function is more representative of the underlying behavior of the algorithm. Nevertheless, recall that experimental data should be interpreted as a lower bound of the worst-case complexity; considering this, new curves are fit subject to the constraint that predicted values be greater or equal than experimental data (i.e., not contradicting the empirical data by producing underestimate predictions). The result is shown in Fig. 2. In this case, the coefficients are $a = 0.0972$ and $b = 8.8484$ for the parabola, and $a = 2.6604$ and $b = 1.4182$ for the power function. The model measures are clearly better for the power function ($s_e = 54.2107$ and $R^2 = 0.99020$) than for the parabola ($s_e = 65.9746$ and $R^2 = 0.98548$).

Fig. 2 here

This analysis can be complemented by studying the growth trend of the empirical data. To do so, we consider a number of candidate functions, $f_1(n), \cdots, f_k(n)$. We then compute the ratio between the empirical data and the values provided by each candidate function. Finally, we calculate a linear interpolation to the obtained data. The rationale behind this is that the better the match between the empirical data and the candidate function, the closer the slope of the linear interpolation will be to zero, i.e., the ratio will tend toward a constant value.

Fig. 3 here

Fig. 3 is an interesting way of displaying this information. It shows the results of this analysis for the following candidate solutions: $f_1(n) = n^2$, $f_2(n) = n \log n$, $f_3(n) = n^{3/2}$, $f_4(n) = n^{\sqrt{2}}$, and $f_5(x) = n^{\phi}$ where $\phi = (1+\sqrt{5})/2 \approx 1.6180$. The obtained data are conclusive, f_1 clearly grows faster than the empirical data. The same holds for f_5. On the contrary, f_2 has a lower growth rate than experimental data. The exact function lies between f_3 and f_4, respectively upper and lower bound of the empirical data. Again, this represents an excellent agreement with theory (recall the $O(n^{3/2})$ optimal result). Notice also that this analysis is consistent with the previous results for BubbleSort (see Fig. 3–right), indicating an excellent match for f_1.

3 Discussion and Future Work

The proposed approach seems promising. For any problem of finite-size, the analysis of algorithms depicted above can provide a useful lower-bound on the worst-case complexity. Notice that by no means we are stating that our work suggests the *"end of proofs"* era. On the contrary, we strongly believe that there is nothing that can replace an elegant proof relating a problem with a worst-case scenario. However, we hope that this mixed evolutionary-statistical analysis can be a positive contribution, particularly when other approaches constitute a hard task for the practitioner and/or the researcher. In this sense, this analysis may assist in this task, providing hard-to-solve instances which could be analyzed and further revised to develop a formal worst-case asymptotic analysis following standard mathematical methods.

Some guidelines for future work are:

(i) Analyze the CPU-time rather than the number of elemental operations, similarly to [2]. This approach could be useful were the nature of these elemental operations difficult to disentangle.

(ii) We plan to apply this approach to *vis-à-vis* algorithmic comparisons as well. The addressed question would be: "how badly can algorithm A perform with respect to algorithm B?", i.e., seeking instances being hard for A but easy for B. This is very different from identifying worst-case scenarios for individual algorithms, and much harder to approach analytically.

(iii) We also intend to use this approach to generate large hard instances, a very useful resource for algorithmic benchmarks. We hypothesize that for some problems it can be possible to identify relevant traits correlated with the hardness of solving an instance. Were this the case, it could be possible to scale-up the instances generated using the presented approach.

Acknowledgements

The first author is partially supported by CICYT under grant TIC1999-0754-C03. The second author is supported by CNPq under Proj. 52.1100/01-1.

References

[1] Th. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.

[2] S. Chakraborty and P.P. Choudhury. A statistical analysis of an algorithm's complexity. *Applied Mathematics Letters*, 13:121–126, 2000.

[3] L.D. Chambers, editor. *Practical Handbook of Genetic Algorithms: Vol.1 Applications.* CRC Press, Boca Ratón FL, 1995.

[4] C. Cotta and J.M. Troya. Genetic forma recombination in permutation flowshop problems. *Evolutionary Computation*, 6(1):25–44, 1998.

[5] B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 143–150, San Mateo CA, 1991. Morgan Kaufmann.

[6] R. Sedgewick. *Algorithms in C++.* Addison-Wesley, 1992.

[7] D.L. Shell. A high-speed sorting procedure. *Communications of the ACM*, 2:30–32, 1959.

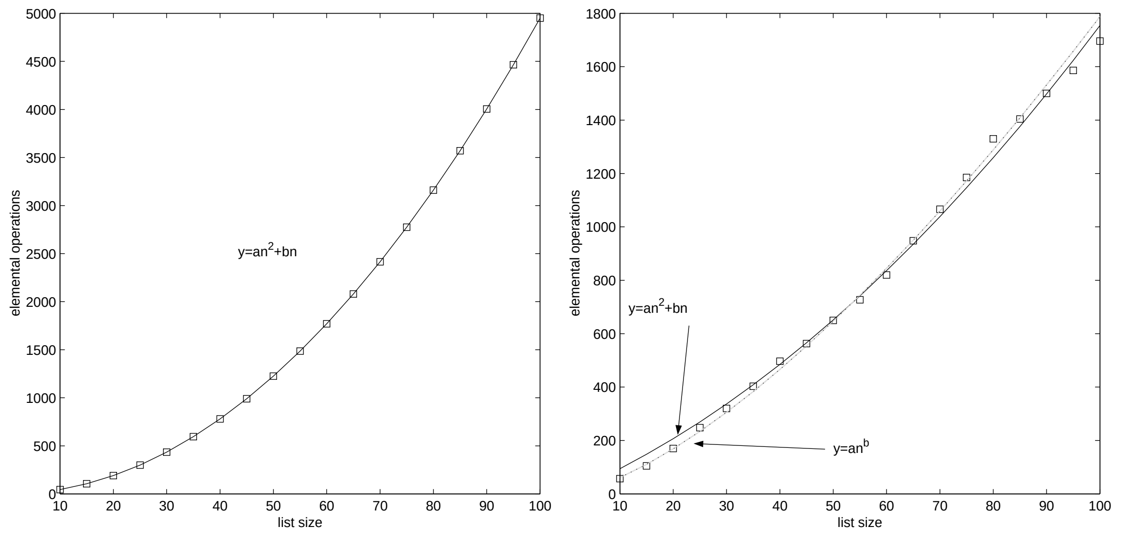

Figure 1: Results obtained by the EA applied to BubbleSort (left) and ShellSort (right). The squares represents the experimental data, the solid line is a fit to $an^2 + bn$, and the dotted line (only in the right figure) is a fit to an^b.

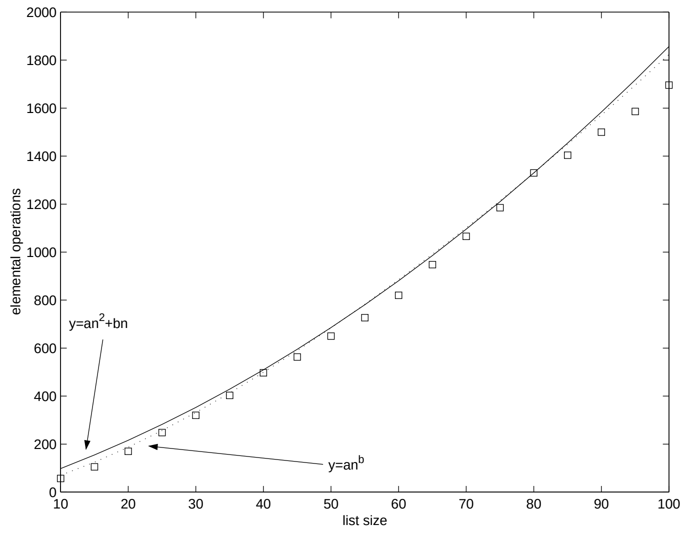

Figure 2: Fitting experimental data for ShellSort to $an^2 + bn$ (solid line) and to an^b (dotted line). Fitted curves are constrained to bound experimental data from above.

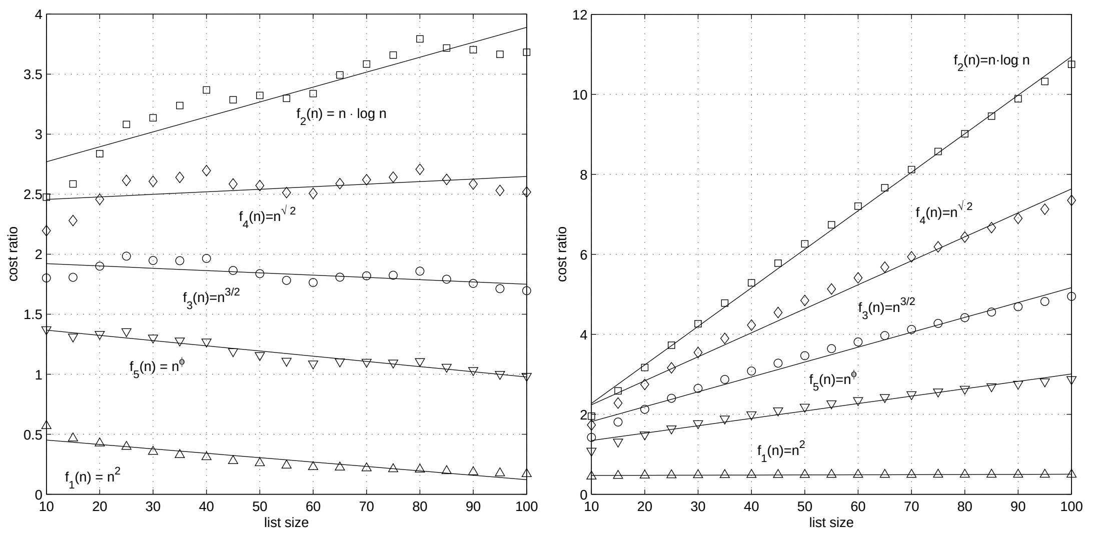

Figure 3: Trend analysis of the cost growth for the empirical data obtained by the EA for ShellSort (left) and BubbleSort (right). A linear interpolation of ratio between experimental data and some tentative cost functions is shown.