Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 36 (2014) 185 – 191

Procedia Computer Science

Complex Adaptive Systems, Publication 4
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2014-Philadelphia, PA

Computational Complexity Measures for Many-objective Optimization Problems

David M. Curry[a,]*, Cihan H. Dagli[a]

[a]Department of Engineering Management and Systems Engineering, Missouri University of Science and Technology, Rolla, Missouri 65409

Abstract

Multi-objective Optimization Problems (MOPs) are commonly encountered in the study and design of complex systems. Pareto dominance is the most common relationship used to compare solutions in MOPs, however as the number of objectives grows beyond three, Pareto dominance alone is no longer satisfactory. These problems are termed "Many-Objective Optimization Problems (MaOPs)". While most MaOP algorithms are modifications of common MOP algorithms, determining the impact on their computational complexity is difficult. This paper defines computational complexity measures for these algorithms and applies these measures to a Multi-Objective Evolutionary Algorithm (MOEA) and its MaOP counterpart.

Keywords: multi-objective optimization problem; MOP; many-objective optimization problem; MaOP; computational complexity; multi-objective evolutionary algorithm; MOEA

1. Introduction

Informally, computational complexity is ultimately concerned with the time and space (resources) required to solve a given problem. While time is often the primary concern, space should be considered as well. Typically, the

* Corresponding author. Tel.: +1-636-699-0997
E-mail address: dmcfh3@mst.edu

doi:10.1016/j.procs.2014.09.077

resources are generically categorized as CPUs (number of independent computational units), primary storage (amount of memory/RAM), secondary storage (disk, cloud, etc.). In the case of multiple CPUs, the topology interconnecting them is also important. Regarding time, it may be measured either in wall-clock time which is pragmatic but machine dependent or by the number of operations required. The standard time complexity measures used in algorithm analysis are framed in iterations which is itself a simplified measure of the number of operations required.

Tractability is an important concept that is related to computational complexity. Computational problems are divided into two categories: tractable and intractable. Notionally, tractable problems can be thought of as possessing a degree of complexity that is solvable and intractable problems as possessing a degree of complexity that is not solvable. Many optimization problems are intractable, so when solving these problems, it is generally not possible to solve them exactly. When speaking of a solution to such a problem, we are actually speaking of solving either an approximation to the actual problem or finding a local optimum rather than the global optimum.

Single objective optimization problems, generally speaking, are intractable time-wise but have finite space (memory) requirements. Multi-objective optimization problems, on the other hand, are intractable both with regards to time and space. Intractability will be dealt with in more detail later, but for now it sufficient to note that for the current purpose, spatial complexity, not temporal complexity creates the essential differences between single and multiple objective algorithms. Many objective optimization problems (MaOP) are a relatively new classification of multiple objective algorithms. The distinction is necessary because as the number of objectives grows, the use of Pareto dominance, which is the corner-stone of MOP algorithms, becomes ineffective. This is because solutions in these higher order spaces no longer dominate one another and therefore the set of optimal solutions becomes, in effect, the entire solution space. The number of objectives where this occurs is problem dependent, but usually begins when there are four to six objectives[1]. Since there are active research fields, such as systems architecting, where the number of objectives routines exceed this threshold, MaOPs are worthy of commiserate attention. Evolutionary algorithms (EAs) will be the focus due to their wide use and applicability to MOPs and MaOPs.

In order to effectively find solutions for MaOPs, the use of Pareto dominance to determine the optimal set must either be modified or abandoned altogether. The effects of this on algorithmic complexity will be explored and complexity measures will be set forth. In order to achieve this, a basic review of complexity measures focusing on "big-oh" notation, NP problems, and intractability is in order along with a review of Pareto dominance since it is fundamental to understanding the difference between single optimization problems, multiple optimization problems, and many optimization problems.

2. Complexity Measures

Computational complexity is usually defined as a function of the size (cardinality) of the input data set. In this case, it is common to speak in terms of the asymptotic growth of an algorithm in terms of the asymptotically dominate term that bounds some aspect of its behavior. When discussing computational complexity, time complexity is assumed unless otherwise stated. For example, if an algorithm requires $6+2n+3n^2$ iterations to finish given any set of input data and n is the cardinality of the input data set, then the algorithm is often called an n^2 algorithm. More accurately, it is $\Theta(n^2)$. An algorithm is said to be $\Theta(f(n))$ if, and only if, it is both $O(f(n))$ and $\Omega(f(n))$. Big-oh (O) denotes the asymptotic upper bound and may be defined as the set of functions $O(f(n)) = \{g(n) : \exists\ c, n_0 \in Z^+ \ni 0 \le g(n) \le cf(n)\ \forall\ n \ge n_0\}$[2]. Big-omega ($\Omega$) denotes the asymptotic upper bound and may be similarly defined as the set of functions $\Omega(f(n)) = \{g(n) : \exists\ c, n_0 \in Z^+ \ni 0 \le cf(n) \le g(n)\ \forall\ n \ge n_0\}$. Big-theta ($\Theta$) would then denote the asymptotically tight bound where it is the set of functions $\Theta(f(n)) = \{ g(n) : \exists\ c_1, c_2, n_0 \in Z^+ \ni 0 \le c_1f(n) \le g(n) \le c_2f(n)\ \forall\ n \ge n_0\}$. Of these, big-oh is the most popularly used as the upper bound is usually of greater interest than the lower bound and an algorithm cannot be described by big-theta unless there exists a function that is both big-oh and big-omega.

A broader way to view computational complexity is to categorize problems and algorithms as either "P" or "NP". Here, P stands for "Polynomial" and NP for "Non-deterministic Polynomial"[3]. Sometimes NP is understood to mean non-polynomial and while not strictly correct, it has never been demonstrated to be false. The difference between P and NP is whether or not the algorithm can be solved in polynomial time or space with a Deterministic Turing Machine (DTM) or if it requires a Non-Deterministic Turing Machine (NDTM). It has been shown that if a problem can be solved in $O(f(n))$ with an NDTM, then it can be solved in $O(2^{f(n)})$ on a DTM. However that is an upper bound and a lower bound has yet to be established. Because the upper bound complexity is exponential for all polynomial $f(n)$, NP can be considered non-polynomial and is often further labeled "intractable". Intractability conveys the notion that a problem is unsolvable due to its time or space complexity, however caution should be applied as this is only the upper bound and the complexity of the average case may be polynomial. One case where this can be seen is Dantzig's simplex method. The simplex method was believed to be a polynomial time algorithm at one time because it usually requires so few iterations, but has now been proven to be NP and intractable[4].

3. Pareto Dominance

Pareto dominance is a way to apply a "less than" or "greater than" relationship to solutions with multiple variables. This relationship is usually defined within the context of whether the problem is a minimization or maximization problem so that the minimum or maximum solutions dominate respectively. For a maximization problem, a solution $s^* = (s_1^*, s_2^*, \ldots, s_n^*)$ is said to dominate $s = (s_1, s_2, \ldots, s_n)$, denoted $s^* \succ s$, if, and only if, $s_i^* \geq s_i, \forall\, i : 1 \leq i \leq n$, and $s_j^* > s_j\ \exists\, j : 1 \leq j \leq n$. For a minimization problem, a solution $s^* = (s_1^*, s_2^*, \ldots, s_n^*)$ is said to dominate $s = (s_1, s_2, \ldots, s_n)$, denoted $s^* \prec s$, if, and only if, $s_i^* \leq s_i, \forall\, i : 1 \leq i \leq n$, and $s_j^* < s_j\ \exists\, j : 1 \leq j \leq n$.

Pareto dominance is necessarily employed weakly in MOPs because it cannot be assumed that one solution would dominate all others. In fact, it is possible for no solution to dominate any other. Therefore, the optimal set is not constructed of the Pareto dominate solutions, since there may not be any, but rather it consists of the set of all non-dominated solutions. As a simple illustration of this, consider a two objective problem where one objective is equal to the negative of the other. If one objective is increased, then the other is decreased, therefore no solution can dominate another and furthermore, every solution is non-dominated and optimal.

4. Optimization Problems

An optimization problem seeks the solution that minimizes or maximizes some objective (single objective optimization problem) or objectives (multiple objective optimization problem) while satisfying a possible set of constraints. The objective is represented as a function and the solutions are comprised of the decision variables that are the parameters of the objective function. The standard form for a constrained single objective optimization problem is

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}),\ \mathbf{x} = (x_1, x_2, \ldots, x_n) \\ \text{subject to} & g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, n_g \\ & h_j(\mathbf{x}) \leq 0,\ j = 1, 2, \ldots, n_h \\ & x_k \in \mathrm{dom}(x_k),\ k = 1, 2, \ldots, n \end{array}$$

where $f(\mathbf{x})$ is the is the objective function, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are the decision variables, $g_i(\mathbf{x}) \leq 0,\ i = 1, 2, \ldots, n_g$ and $h_j(\mathbf{x}) \leq 0,\ j = 1, 2, \ldots, n_h$ are the constraint functions, $\mathrm{dom}(x_k),\ k = 1, 2, \ldots, n$ are the constraint sets, n is the number of decision variables, n_g is the number of inequality constraints, and n_h is the number of equality constraints. This can be extended to the multiple objective problem by replacing $f(\mathbf{x})$ with $f_m(\mathbf{x})$, $m = 1, 2, \ldots, p$, where p is the number of objectives. The problem is stated as a minimization problem, but without loss of generality as any maximization problem may be formulated as a minimization problem.

There are a number of considerations regarding optimization problems. Some important ones are whether it is constrained or unconstrained, the domain of the each decision variable (reals, integers, etc.), the number of objectives, and if the objective function(s) are differentiable. To be as widely applicable as possible, the most general case of each will be taken. To this end, optimization problems will be treated as multiple objective, constrained, and non-differentiable.

5. Evolutionary Algorithms

Evolutionary Algorithms mimic the concept of biological evolution. An optimal solution is found by performing biologically inspired operations upon a population and thus creating the next generation of individuals. These individuals are evaluated using fitness functions that are the objective functions defining an optimization problem. Based upon fitness and other criteria, some individuals are removed from the population and the process is repeated until a stopping criteria is meet and the more fit individual(s) are the optimal solutions(s). Individuals are represented by a chromosome that encodes the decision variables. Careful encoding can be used to define chromosomes that adhere to the domain of each decision variable and sometimes even to satisfy certain constraints.

A generic EA has the following form[5]:

```
Generic Evolutionary Algorithm
$t \leftarrow 0$                                                Generation counter
$C_0$                                                           Initial population
while stopping condition(s) not true do                         Create new generation
    Evaluate fitness, $f(\mathbf{x})$, of each individual $\mathbf{x} \in C_t$
    Perform reproduction to create offspring
    Select the new population $C_{t+1}$
    $t \leftarrow t + 1$                                        Advance to new generation
end while
```

While this is straight-forward for a fitness function with a single objective, it is not so clear in the case of multiple objectives. Evolutionary algorithms that handle multiple objectives are termed Multiple Objective Evolutionary Algorithms (MOEAs). MOEAs typically rely on Pareto dominance to determine a non-dominated set of individuals to retain or use for reproduction. Many methods have been employed to make these determinations so that exploration and exploitation are achieved (this is sometimes referred to as "progress"[6]). To demonstrate the variety of methods that are possible, here is a partial list of current MOEAs[7]:

- Vector Evaluated GA (VEGA)
- Lexicographic Ordering GA
- Vector Optimized Evolution Strategy (VOES)
- Weight-Based GA (WBGA)
- Multiple Objective GA (MOGA)
- Niched Pareto GA (NPGA, NPGA 2)
- Nondominated Sorting GA (NSGA, NSGA-II)
- Distance-based Pareto GA (DPGA)
- Thermodynamical GA (TDGA)
- Strength Pareto Evolutionary Algorithm (SPEA, SPEA2)
- Multi-Objective Messy GA (MOMGA-I,II,III)
- Pareto Archived ES (PAES)
- Pareto Envelope-based Selection Algorithm (PESA, PESA II)
- Micro GA-MOEA (μGA, μGA2)
- Multi-Objective Bayesian Optimization Algorithm (mBOA)

While these algorithms are designed for MOPs, they may not work well for MaOPs because of the inadequacy of Pareto dominance when there is a large number of objectives present. Taking this into consideration, here are some modifications to Pareto dominance to be incorporated directly into other MOEAs that have been suggested:

- Fuzzy Dominance[8]
- Corner Sort[9]
- Objective Reduction[10]
- Hybrid Framework[11]

Here are a few MaOP specific algorithms that have been advanced:

- Pareto Corner Search Evolutionary Algorithm (PCSEA)[12]
- Borg[13]
- Fuzzy Dominance SPEA2 (FD-SPEA2)[8]
- Multiobjective Evolutionary Algorithm (DMOEA)[14]
- Nondominated Sorting GA (NSGA-III)[15]
- Multiobjective Evolutionary Algorithm/Decomposition (MOEA/D)[16]

These algorithms usually attempt to assist exploration and exploitation (progress) by either modifying the definition of Pareto dominance or by adding additional steps.

The advantage of EAs is that they do not require the objective (fitness) function to be convex, linear, or differentiable (i.e. they are a non-gradient method). The downside is that it is difficult to explore the solution space without a gradient. Therefore, when any of these properties are met, EAs are inefficient compared to those algorithms that can utilize the additional information.

6. MOEA Complexity

The space complexity of MOEAs is simple: $\Theta(n_p)$, where n_p is the population size. This, of course, assumes a fixed population size, but that is expected even with algorithms using external archiving. With other, non-evolutionary, types of algorithms, the space complexity can be NP since the non-dominated Pareto set can be uncountably infinite. However, EAs, as a rule, only maintain a specific population size, so in their case the space complexity is linear (P).

The time complexity of MOEAs is more complicated. The upper bound time complexity of an MOEA is dictated by the length of the given chromosome because it can be exhaustively searched in $\Theta(2^{nc})$ time, where n_c is the number of bits required to encode the chromosome. However, since searching exhaustively is guaranteed to have exponential (NP) time complexity (lower bound as well as upper), it is too inefficient to be practical. Even though no MOEA, by definition, employs an exhaustive search of all the possible chromosome permutations, this allows a worst-case bound of $O(2^{nc})$ to be set for the entire class with the exception of specific algorithms proven to have a better bound.

Like the simplex method mentioned earlier, even though the upper bound is NP, the average case and lower bound may be P. Some of the more common MOEAs have complexity that appears to be P. For example, slower MOEAs like NSGA[17] and SPEA[18] are $O(n_o n_p^3)$, while faster ones like NSGA-II[19], SPEA2[20], and PAES[21] are $O(n_o n_p^2)$, where n_o is the number of objectives. The catch is that the complexity given here is the computational complexity involved for advancing a single generation of the population, not the algorithm's aggregate complexity. The reason why the computational complexity is often stated with respect to a single generation is that it allows algorithms to be compared when the convergence rate of the overall algorithm is unknown.

In order to calculate the actual computational complexity of an MOEA, it is necessary to know both the complexity for each generation and the number of generations. Although the convergence of EAs using at least the operations of reproduction and mutation combined with elitism has been proven[22,23,24], the rate of convergence has not. Therefore, we can say that the faster algorithms such as NSGA-II, SPEA2, and PAES are $O(n_g n_o n_p^2)$, where n_g is the number of generations. Depending upon the stopping criteria used, n_g can have any complexity from constant to NP. It should be noted that when n_g is not constant or otherwise limited, it is a function of the chromosome length, n_c.

Since we now have defined the complexity of three algorithms representative of the fastest general purpose MOEAs, we now turn our attention to those specifically designed for MaOPs. We shall examine two such algorithms, FD-SPEA2 and NSGA-III and compare them to their conventional MOEA counterparts, SPEA2 and NSGA-II, respectively. This information can then be used to understand the impact that solving for a large number of objectives has on multiple objective optimization problems.

FD-SPEA2 is a hybrid method that replaces the fitness assignment with a fuzzy fitness assignment and a fuzzy ranking assignment. The fuzzy fitness assignment process needs to assess each individual of the population with regard every other individual for each objective and is therefore $O(n_g n_o n_p^2)$. The fuzzy ranking assignment process may need to execute the fuzzy fitness assignment process up to n_p times, making the entire process $O(n_g n_o n_p^3)$. Since the fuzzy ranking assignment process is performed once per generation, the overall complexity is $O[n_g(n_o n_p^3 + n_o n_p^2)] = O(n_g n_o n_p^3)$.

NSGA-III is nearly identical to NSGA-II except for the selection process. The NSGA-III selection process utilizes reference points to apply the selection pressure that a non-dominated Pareto approach lacks. The effect of this approach on the computational complexity is minimal as NSGA-III is $O[\max(n_g n_p^2 \log^{no-2} n_p, n_g n_o n_p^2)]$[15].

7. Conclusion

Only the effect of a large number of objectives on computational complexity when multiple objective evolutionary algorithms are employed as optimizers has been analyzed. Incorporating other optimization approaches in the analysis would certainly strengthen this work, and is an obvious source of future work in this area. However, the results from the MOEAs should be indicative of the field as a whole since they are among the state-of-the-art in multiple objective optimization. Given this, the effect on computational complexity varies when incorporating an approach that is designed for the additional challenges inherent to MaOPs that don't exist in other MOPs. One MaOP MOEA, FD-SPEA2, is significantly slower than its MOP counterpart, SPEA2. However, depending on the size of the population relative to the number of objectives, NSGA-III can have the same computational complexity as its MOP parent, NSGA-II. We can therefore conclude that many objective optimization is essentially no more complex than conventional multiple objective optimization when presented with a suitable algorithm.

References

1. H. Ishibuchi, N. Tsukamoto, Y. Hitotsuyanagi, Y. Nojima, Effectiveness of scalability improvement attempts on the performance of nsga-ii for many-objective problems, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO '08, ACM, New York, NY, USA, 2008, pp. 649–656. doi:10.1145/1389095.1389225.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, McGraw-Hill Higher Education, 2001.
3. M. R. Garey, D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
4. D. G. Luenberger, Y. Ye, Linear and Nonlinear Programming, 3rd Edition, International Series in Operations Research and Management Science, Springer Science+Business Media, LLC, New York, NY, USA, 2008.
5. A. P. Engelbrecht, Computational Intelligence: An Introduction, 2nd Edition, John Wiley & Sons Ltd., Chichester, West Sussex, England, 2007.
6. D. A. Van Veldhuizen, Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations, Ph.D. thesis, Wright Patterson AFB, OH, USA, aAI9928483 (1999).
7. C. A. Coello Coello, G. B. Lamont, D. A. V. Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation), 2nd Edition, Genetic and Evolutionary Computation, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
8. Z. He, G. G. Yen, J. Zhang, Fuzzy-based pareto optimality for many-objective evolutionary algorithms, Evolutionary Computation, IEEE Transactions on 18 (2) (2014) 269–285. doi:10.1109/TEVC.2013.2258025.
9. H. Wang, X. Yao, Corner sort for pareto-based many-objective optimization, Cybernetics, IEEE Transactions on 44 (1) (2014) 92–102. doi:10.1109/TCYB.2013.2247594.
10. D. Saxena, J. Duro, A. Tiwari, K. Deb, Q. Zhang, Objective reduction in many-objective optimization: Linear and nonlinear algorithms, Evolutionary Computation, IEEE Transactions on 17 (1) (2013) 77–99. doi:10.1109/TEVC.2012.2185847.
11. K. Sindhya, K. Miettinen, K. Deb, A hybrid framework for evolutionary multi-objective optimization, Evolutionary Computation, IEEE Transactions on 17 (4) (2013) 495–511. doi:10.1109/TEVC.2012.2204403.
12. H. K. Singh, A. Isaacs, T. Ray, A pareto corner search evolutionary algorithm and dimensionality reduction in many-objective optimization problems, Evolutionary Computation, IEEE Transactions on 15 (4) (2011) 539–556. doi:10.1109/TEVC.2010.2093579.

13. D. Hadka, P. Reed, Borg: An auto-adaptive many-objective evolutionary computing framework, Evol. Comput. 21 (2) (2013) 231–259. doi:10.1162/EVCO a 00075.
14. X. Zou, Y. Chen, M. Liu, L. Kang, A new evolutionary algorithm for solving many-objective optimization problems, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 38 (5) (2008) 1402–1412. doi:10.1109/TSMCB.2008.926329.
15. K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part i: Solving problems with box constraints (2013). doi:10.1109/TEVC.2013.2281534.
16. Q. Zhang, H. Li, Moea/d: A multiobjective evolutionary algorithm based on decomposition, Evolutionary Computation, IEEE Transactions on 11 (6) (2007) 712–731. doi:10.1109/TEVC.2007.892759.
17. N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, Evol. Comput. 2 (3) (1994) 221–248. doi:10.1162/evco.1994.2.3.221.
18. E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms—a comparative case study, in: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, PPSN V, Springer-Verlag, London, UK, 1998, pp. 292–304.
19. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, Evolutionary Computation, IEEE Transactions on 6 (2) (2002) 182–197. doi:10.1109/4235.996017.
20. E. Zitzler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, in: K. C. Giannakoglou, D. T. Tsahalis, J. Periaux, K. D. Papailiou, T. Fogarty (Eds.), Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, International Center for Numerical Methods in Engineering, Athens, Greece, 2001, pp. 95–100.
21. J. Knowles, D. Corne, The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation, in: Evolutionary Computation, 1999. Proceedings of the 1999 Congress on, Vol. 1, 1999, pp. 98–105. doi:10.1109/CEC.1999.781913.
22. T. E. Davis, J. C. Principe, A markov chain framework for the simple genetic algorithm, Evolutionary Computing 1 (3) (1993) 269–288. doi:10.1162/evco.1993.1.3.269.
23. H. Dawid, A markov chain analysis of genetic algorithms with a state dependent fitness function, Complex Systems 8 (1994) 407–417.
24. G. Rudolph, Convergence analysis of canonical genetic algorithms, Neural Networks, IEEE Transactions on 5 (1) (1994) 96–101. doi:10.1109/72.265964.