THEORETICAL FRAMEWORK FOR COOPERATION AND COMPETITION IN EVOLUTIONARY COMPUTATION

Eugene Eberbach*
Dept. of Eng. and Science, Rensselaer Polytechnic Institute, 275 Windsor Street, Hartford, CT 06120, USA

Mark Burgin
Dept. of Mathematics, University of California, 405 Hilgard Avenue, Los Angeles, CA 90095, USA

Keywords: Combinatorial optimization, evolutionary computation, complexity theory, multiobjective optimization, search theory, cooperation and competition, super-recursive algorithms, Evolutionary Turing Machine.

Abstract: In the paper the theoretical framework for cooperation and competition of coevolved population members working toward a common goal is presented. We use a formal model of Evolutionary Turing Machine and its extensions to justify that in general evolutionary algorithms belong to the class of super-recursive algorithms. Parallel and Parallel Weighted Evolutionary Turing Machine models have been proposed to capture properly cooperation and competition of the whole population expressed as an instance of multiobjective optimization.

1 INTRODUCTION

In this paper, we study the following problems for the finite types of evolutionary computations: completeness, optimality, search optimality, total optimality and decidability for single and multiple cooperating or competing individuals. In particular, we concentrate our attention on the problem of cooperation and competition of coevolved individuals in population expressed in a natural way as an instance of multiobjective optimization.

The paper is organized as follows. In section 2 we give a short primer on problem solving. Section 3 presents Evolutionary Algorithms and an Evolutionary Turing Machine as a formal model of evolutionary computation. A formal model for cooperating and competitive population agents trying to achieve a common goal is developed in Section 4. Section 5 contains conclusions.

* Research supported initially by ONR under grant N00014-06-1-0354. On leave from University of Massachusetts Dartmouth, eeberbach@umassd.edu.

2 EVOLUTION AND PROBLEM SOVING

All algorithms are divided into three big classes (Burgin, 2005): *subrecursive*, *recursive*, and *super-recursive*.

Algorithms and automata that have the same computing/accepting power (cf., (Burgin, 2005a)) as Turing machines are called *recursive*. Examples are partial recursive functions or random access machines.

Algorithms and automata that are weaker than Turing machines are called *subrecursive*. Examples are finite automata, context free grammars or push-down automata.

Algorithms and automata that are more powerful than Turing machines are called super-recursive. Examples are inductive Turing machines, Turing machines with oracles or finite-dimensional machines over the field of real numbers.

The performance of search algorithms can be evaluated in four ways (see, e.g., (Russell and Norvig, 1995) capturing three criteria: whether a solution has been found, its quality and the amount of resources used to find it.

Definition 2.1. (*Completeness, optimality, search optimality, and total optimality*) We say that the search algorithm is

- *Complete* if it guarantees reaching a terminal state/solution if there is one.
- *Optimal* if it finds the solution with the optimal value of its objective function.
- *Search Optimal* if it finds the solution with the minimal amount of resources used (e.g., minimal time or space complexity).
- *Totally Optimal* if it finds the solution both with the optimal value of its objective function and with the minimal amount of resources used.

Let $\boldsymbol{R}$ be the set of all real numbers and $\boldsymbol{R}^+$ be the set of all non-negative real numbers.

Definition 2.2. (*Problem solving as a multiobjective optimization problem)*

Given an objective function f: $\mathbf{A} \times X \rightarrow \boldsymbol{R}^+$, problem-solving can be considered as a *multiobjective minimization problem* to find $A^* \in \mathbf{A}_F$ and $x^* \in X_F$ such that

$$f(A^*, x^*)= \min\{f_1(f_2(A), f_3(x)), A \in \mathbf{A}, x \in X\}$$

where f_3 is a *problem-specific objective function*, f_2 is a *search algorithm objective function*, and f_1 is an *aggregating function* combining f_2 and f_3.

3 EVOLUTIONARY TURING MACHINES

Definition 3.1. A generic *evolutionary algorithm (EA)* can be described in the form of the functional equation (recurrence relation) working in a simple iterative loop in discrete time t, called generations, t = 0, 1, 2,... (Fogel, 1995, Michalewicz and Fogel, 2004):

$$X[t+1] = s\ (v\ (X[t])), \text{ where}$$

- $X[t] \subseteq X$ is a *population* under a representation consisting of one or more individuals from the set X (e.g., fixed binary strings for genetic algorithms (GAs), Finite State Machines for evolutionary programming (EP), parse trees for genetic programming (GP), vector of reals for evolution strategies (ES)),
- s is a *selection* operator (e.g., *truncation, proportional, tournament*),
- v is a *variation* operator (e.g., variants of *mutation* and *crossover*),
- $X[0]$ is an *initial* population,
- $F \subseteq X$ is the set of *final* populations satisfying the *termination condition* (goal of evolution). The desirable termination condition is the optimum in X of the *fitness* function f: $X \rightarrow \boldsymbol{R}$, which is extended to the *fitness* function $f(X[t])$ of the best individual in the population $X[t] \in F$, where f is defined typically in the domain of nonnegative real numbers. In many cases, it is impossible to achieve or verify this optimum. Thus, another stopping criterion is used (e.g., the maximum number of generations, the lack of progress through several generations.).

Definition 3.1 is applicable to all typical EAs, including GA, EP, ES, GP.

Formally, an evolutionary algorithm looking for the optimum of the fitness function violates some classical requirements of recursive algorithms. If its termination condition is set to the optimum of the fitness function, it may not terminate after a finite number of steps. To fit it to the old "algorithmic" approach, an artificial (or somebody can call it pragmatic) stop criterion has had to be added (see e.g., (Michalewicz, 1996; Koza, 1992)). The evolutionary algorithm, to remain recursive, has to be stopped after a finite number of generations or when no visible progress is observable. Naturally, in a general case, Evolutionary Algorithms are instances of super-recursive algorithms.

Now, we define a formal algorithmic model of Evolutionary Computation - an *Evolutionary Turing Machine* (Eberbach, 2005).

Definition 3.2. An *evolutionary Turing machine* (*ETM*) $E = \{ \text{TM}[t]; t = 0, 1, 2, 3, ... \}$ is a series of (possibly infinite) Turing machines TM[t] each working on population $X[t]$ in generations t = 0, 1, 2, 3, ... where

- each $\delta[t]$ transition function (rules) of the Turing Machine TM[t] represents (encodes) an evolutionary algorithm that works with the population $X[t]$, and evolved in generations 0, 1, 2, ... , t,
- only generation 0 is given in advance, and any other generation depends on its predecessor only, i.e., the outcome of the generation t = 0, 1, 2, 3, ... is the population $X[t + 1]$ obtained by

applying the recursive variation *v* and selection *s* operators working on population *X*[*t*],

- (TM[0], *X*[0]) is the initial Turing Machine operating on its input - an initial population *X*[0],
- the goal (or halting) state of ETM *E* is represented by any population *X*[*t*] satisfying the termination condition. The desirable termination condition is the optimum of the fitness performance measure *f*(*x*[*t*]) of the best individual from the population *X*[*t*].
- When the termination condition is satisfied, then the ETM *E* halts (*t* stops to be incremented), otherwise a new input population *X*[*t* + 1] is generated by TM[*t* + 1].

In this model, both variation *v* and selection *s* operators are realized by Turing machines. So, it is natural that the same Turing machine computes values of the fitness function *f*. This brings us to the concept of a weighted Turing machine.

Definition 3.3. A *weighted Turing machine* (*T* , *f*) computes a pair (*x*, *f*(*x*)) where *x* is a word in the alphabet of *T* and *f*(*x*) is the value of the evaluation function *f* of the machine (*T* , *f*).

It is necessary to remark that only in some cases it is easy to compute values of the fitness function *f*. Examples of such situations are such fitness functions as the length of a program or the number of parts in some simple system. However, in many other cases, computation of the values of the fitness function *f* can be based on a complex algorithm and demand many operations. For instance, when the optimized species are programs and the fitness function *f* is time necessary to achieve the program goal, then computation of the values of the fitness function *f* can demand functioning or simulation of programs generated in the evolutionary process. We encounter similar situations when optimized species are computer chips or parts of plane or cars. In this case, computation of the values of the fitness function *f* involves simulation.

Weighted computation realized by weighted Turing machines allows us to extend the formal algorithmic model of Evolutionary Computation defining a *Weighted Evolutionary Turing Machine*.

Definition 3.4. A *weighted evolutionary Turing machine* (*WETM*) *E* = { TM[*t*]; *t* = 0, 1, 2, 3, ... } is a series of (possibly infinite) weighted Turing machines TM[*t*] each working on population *X*[*t*] in generations *t* = 0, 1, 2, 3, ... where

- each δ[*t*] transition function (rules) of the weighted Turing machine TM[*t*] represents (encodes) an evolutionary algorithm that works with the population *X*[*t*], and evolved in generations 0, 1, 2, ... , *t*,
- only generation 0 is given in advance, and any other generation depends on its predecessor only, i.e., the outcome of the generation *t* = 0, 1, 2, 3, ... is the population *X*[*t* + 1] obtained by applying the recursive variation *v* and selection *s* operators working on population *X*[*t*] and computing the fitness function *f*,
- (TM[0], *X*[0]) is the initial weighted Turing Machine operating on its input - an initial population *X*[0],
- the goal (or halting) state of WETM *E* is represented by any population *X*[*t*]) satisfying the termination condition. The desirable termination condition is the optimum of the fitness performance measure *f*(*x*[*t*]) of the best individual from the population *X*[*t*].
- When the termination condition is satisfied, then the WETM *E* halts (*t* stops to be incremented), otherwise a new input population *X*[*t* + 1] is generated by TM[*t* + 1].

The concept of a universal automaton/algorithm plays an important role in computing and is useful for different purposes. The construction of universal automata and algorithms is usually based on some codification (symbolic description) **c**: **K** → *X* of all automata/algorithms in **K**.

Definition 3.5. An automaton/algorithm *U* is *universal* for the class **K** if given a description **c**(*A*) of an automaton/algorithm *A* from **K** and some input data *x* for it, *U* gives the same result as *A* for the input *x* or gives no result when *A* gives no result for the input *x*.

This leads us immediately, following Turing's ideas, to the concept of the universal Turing machine and its extensions - a *Universal Evolutionary Turing Machine* and *Weighted Evolutionary Turing Machine*. We can define a Universal Evolutionary Turing Machine as an abstraction of all possible ETMs, in the similar way, as a universal Turing machine has been defined, as an abstraction of all possible Turing machines.

Definition 3.6. A *universal evolutionary Turing machine* (*UETM*) is an ETM *EU* with the optimization space $Z = X \times I$. Given a pair (**c**(*E*), *X*[0]) where *E* = { TM[*t*]; *t* = 0, 1, 2, 3, ... } is an

ETM and $X[0]$ is the start population, the machine *EU* takes this pair as its input and produces the same population $X[1]$ as the Turing machine TM[0] working with the same population $X[0]$. Then *EU* takes the pair (**c**(*E*), $X[1]$) as its input and produces the same population $X[2]$ as the Turing machine TM[1] working with the population $X[1]$. In general, *EU* takes the pair (**c**(*E*), $X[t]$) as its input and produces the same population $X[t + 1]$ as the Turing machine TM[t] working with the population $X[t]$ where $t = 0, 1, 2, 3, \ldots$.

In other words, by a *Universal Evolutionary Turing Machine* (*UETM*) we mean such ETM *U* that on each step takes as the input a pair (**c**(TM[t]), $X[t]$) and behaves like ETM *E* with input $X[t]$ for $t = 0, 1, 2, \ldots$. UETM *U* stops when ETM *E* stops.

Definition 3.6 gives properties of but does not imply its existence. However, as in the case of Turing machines, we have the following result.

Theorem 3.1. (Eberbach, 2005). In the class of all evolutionary Turing machines, there is a universal evolutionary Turing machine.

Definition 3.7. A *universal weighted evolutionary Turing machine* (*UWETM*) is an WETM *EU* with the optimization space $Z = X \times I$. Given a pair (**c**(*E*), $X[0]$) where $E = \{ \text{TM}[t]; t = 0, 1, 2, 3, \ldots \}$ is an WETM and $X[0]$ is the start population, the machine *EU* takes this pair as its input and produces the same population $X[1]$ as the weighted Turing machine TM[0] working with the same population $X[0]$. Then *EU* takes the pair (**c**(*E*), $X[1]$) as its input and produces the same population $X[2]$ as the weighted Turing machine TM[1] working with the population $X[1]$. In general, *EU* takes the pair (**c**(*E*), $X[t]$) as its input and produces the same population $X[t + 1]$ as the weighted Turing machine TM[t] working with the population $X[t]$ where $t = 0, 1, 2, 3, \ldots$.

This definition gives properties of but does not imply its existence.

Theorem 3.2. In the class of all weighted evolutionary Turing machines with a given recursively computable weight (fitness) function *f*, there is a universal weighted evolutionary Turing machine.

4 COOPERATION AND COMPETITION IN EVOLUTIONARY COMPUTATION

Popular models of distributed intelligent performance (e.g., optimization) are coevolutionary systems (Michalewicz and Fogel, 2004), Particle Swarm Optimization (PSO, also called Swarm Intelligence) (Kennedy et al, 1995), and Ant Colony Optimization (ACO also known as Ant Colony System (ACS)) (Bonabeau et al, 1999).

Coevolution, ant colony optimization and particle swarm optimization seem be potentially the most useful subareas of evolutionary computation for expressing interaction of multiple agents (in particular, to express their cooperation and competition). However, paradoxically in most current applications, these techniques are used to obtain optimal solutions for optimization of single agent behavior (in presence of other agents – members of population), and not for the optimization of group of agents trying to achieve a common goal (represented by joint fitness function). This is primarily because fitness function is optimized for a single individual from the population, and not for the population as a whole.

Now, we define a formal algorithmic model of Evolutionary Computation with cooperation and competition – a *Parallel Evolutionary Turing Machine.*

Definition 4.1. A *parallel evolutionary Turing machine* (*PETM*) $E = \{ \text{TM}_i[t]; t = 0, 1, 2, 3, \ldots; i \in I \}$ consists of a collection of series of (possibly infinite) Turing machines $\text{TM}_i[t]$ each working on population $X[t]$ in generations $t = 0, 1, 2, 3, \ldots$ where

- each $\delta_i[t]$ transition function (rules) of the Turing machine $\text{TM}_i[t]$ represents (encodes) an evolutionary algorithm that works with the whole generation $X[t]$ based on its own fitness performance measure $f_i(x[t])$, and evolved in generations $0, 1, 2, \ldots , t$,
- the whole generation $X[t]$ is the union of all subgenerations $X_i[t]$ obtained by all Turing machines $\text{TM}_i[t - 1]$ that collaborate in generating $X[t]$,
- only the zero generation $X[0]$ is given in advance, and any other generation depends on its predecessor only, i.e., the outcome of the generation $t = 0, 1, 2, 3, \ldots$ is the subgeneration

$X_i[t + 1]$ obtained by applying the recursive variation *v* and selection *s* operators working on the whole generation $X[t]$ and realized by the Turing machine $TM_i[t]$,

- $TM_i[0]$ are the initial Turing machines operating on its input - an initial population $X[0]$,
- the goal (or halting) state of PETM *E* is represented by any population $X[t]$) satisfying the termination condition. The desirable termination condition is the optimum of the unified fitness performance measure $f(X[t])$ of the whole population $X[t]$.
- when the termination condition is satisfied, then the PETM *E* halts (*t* stops to be incremented), otherwise a new input population $X[t + 1]$ is generated by machines $TM_i[t + 1]$.

In a similar way, we define a Parallel Weighted Evolutionary Turing Machine.

Definition 4.2. A *parallel weighted evolutionary Turing machine* (*PWETM*) $E = E = \{ TM_i[t]; t = 0, 1, 2, 3, ...; i \in I \}$ consists of a collection of series of (possibly infinite) Turing machines $TM_i[t]$ each working on population $X[t]$ in generations $t = 0, 1, 2, 3, ...$ where

- each $\delta_i[t]$ transition function (rules) of the Turing machine $TM_i[t]$ represents (encodes) an evolutionary algorithm that works with the whole generation $X[t]$ based on its own fitness performance measure $f_i(x[t])$, and evolved in generations $0, 1, 2, ... , t$,
- the whole generation $X[t]$ is the union of all subgenerations $X_i[t]$ obtained by all Turing machines $TM_i[t - 1]$ that collaborate in generating $X[t]$,
- only the zero generation $X[0]$ is given in advance, and any other generation depends on its predecessor only, i.e., the outcome of the generation $t = 0, 1, 2, 3, ...$ is the subgeneration $X_i[t + 1]$ obtained by applying the recursive variation *v* and selection *s* operators working on the whole generation $X[t]$ and computing the fitness function f_i , and realized by the Turing machine $TM_i[t]$,
- $TM_i[0]$ are the initial Turing machines operating on its input - an initial population $X[0]$,
- the goal (or halting) state of PETM *E* is represented by any population $X[t]$) satisfying the termination condition. The desirable termination condition is the optimum of the unified fitness performance measure $f(X[t])$ of the whole population $X[t]$.

when the termination condition is satisfied, then the PETM *E* halts (*t* stops to be incremented), otherwise a new input population $X[t + 1]$ is generated by machines $TM_i[t + 1]$.

Our models (of PETM and PWETM) are already prepared to handle such situation. It is enough to assume that fitness functions f, f_1, f_2, and f_3 are computed for the whole population (perhaps, consisting of subpopulations), and not for separate individuals from the population only. Let's assume that our population $| x | = p$, i.e., it consists of p individuals or subpopulations. For simplicity, let's consider only individuals (by adding multiple indices, we can consider subpopulations without losing the generality of the approach).

Let $f(M[t],X[t])=f_1(f_2(M[t]),f_3(X[t]))$, where $M[t]=\{M_1[t],\ldots,M_p[t]\}$, $X[t]=\{X_1[t],\ldots,X_p[t]\}$. We define a problem-specific fitness function f_3 for the whole population $f_3(X[t])=f_{13}(f_{31}(X_1[t]),\ldots,f_{3p}(X_p[t]))$, where f_{13} is an aggregating function for $f_{31},\ldots,f_{3p}$, and f_{3j} is a fitness function of the j-th individual x_j, $j =1,\ldots,p$, and an evolutionary algorithm fitness function f_2 for the whole population $f_2(M[t])=f_{12}(f_{21}(M_1[t]),\ldots,f_{2p}(M_p[t]))$, where f_{12} is an aggregating function for $f_{21},\ldots,f_{2p}$, and f_{2j} is a fitness function of the j-th evolutionary algorithm M_j , $j = 1, \ldots, p$, and evolutionary algorithm M_j is responsible for evolution of X_j.

We will present definition for cooperation and competition for generic fitness function *f*. Similar definitions can be provided for fitness functions $f_2$and f_3.

Definition 4.3. (*Cooperation of single individual with population*) We will say that j-th individual x_j *cooperates* in time t with the whole population with respect to a fitness function *f* iff $f[t] > f[t + 1]$ and other's individuals fitness functions are fixed $f_i[t] = f_i[t + 1]$ for $i \neq j$.

Definition 4.4. (*Cooperation of the whole population*) We will say that all population *cooperates* as the whole in time *t* with respect to a fitness function *f* iff $f[t] > f[t + 1]$.

Definition 4.5. (*Competition of single individual with population*) We will say that j-th individual x_j *competes* in time t with the whole population with respect to a fitness function *f* iff $f[t] < f[t+1]$ and other's individuals fitness functions are fixed $f_i[t] = f_i[t + 1]$ for $i \neq j$.

Definition 4.6. (*Competition of the whole population*) We will say that all population *competes* as the whole in time t with respect to a fitness function *f* iff $f[t] < f[t+1]$.

In other words, if individual decreases (increases) fitness function of the whole population then it cooperates (competes) with it. If fitness function of the whole population decreases (increases) then the population exhibits cooperation (competition) as the whole (independently what its individuals are doing). If individual (population) cooperates (competes) for all moments of time, then it is always cooperative (competitive). Otherwise, it may sometimes cooperate, sometimes compete.

Let us consider some problems.

Analysis problem for f_3 : Given $f_1[0],\dots,f_p[0]$ for individuals $x_1[0],\dots,x_p[0]$ from the population $X[0]$ and aggregating function is given. What will be the behavior (emerging, limit behavior) of $f[t]$ for $X[t]$?

Synthesis/design problem for f_3 : Given $f[0]$ for the population $X[0]$. Find corresponding individuals $x_1[0],\dots,x_p[0]$ with $f_1[0],\dots,f_p[0]$ and aggregating function that $f[t]$ will converge to optimum.

Theorem 4.1. (*Optimality of evolutionary computation with cooperating population – sufficient conditions to solve the synthesis problem for f*) For a given evolutionary algorithm *UT*[0] with population *X*[0], if UETM *EU* = { UT[*t*]; *t* = 0, 1, 2, 3, ... } satisfies three conditions

1. the termination condition is set to the optimum of the fitness function $f(X[t])$ with the optimum f^*,
2. search is complete, and
3. population is cooperative all time t = 0, 1, 2, … ,

then UETM *EU* is guaranteed to find the optimum X^* of $f(X[t])$ in an unbounded number of generations t = 0, 1, 2, ... , and that optimum will be maintained thereafter.

Note that cooperation replaces elitism in sufficient condition for convergence of cooperating members of population looking for the optimum of fitness of the whole population and not of the separate individual. There is no surprise: if the whole population competes all the time, then the optimum will not be found and maintained despite completeness.

Theorem 4.2. (*Optimality of evolutionary computation with competing population – inability to solve the synthesis problem for f*) For a given evolutionary algorithm *UT*[0] with population *X*[0], if UETM *EU* = { UT[*t*]; *t* = 0, 1, 2, 3, ... } satisfies three conditions

1. the termination condition is set to the optimum of the fitness function $f(X[t])$ with the optimum f^*,
2. search is complete, and
3. population is competing all time t=0,1,2,…,

then UETM *EU* is not guaranteed to find and maintain the optimum X^* of $f(X[t])$ even in an unbounded number of generations t = 0, 1, 2, ...

If population is sometimes competing, sometimes cooperating, then the optimum sometimes will be found, sometimes not, but the convergence and its maintenance is not guaranteed.

5 CONCLUSIONS

In this paper, we presented a formal model of cooperation and competition for evolutionary computation. We believe that our model constitutes the first formal, much more precise and more generic approach trying to capture the essence of cooperation and competition for evolutionary algorithms. This was possible because of precise formulation on notions of cooperation, competition, completeness, various types of optimization, an extension of the notion of decidability – all of them used in the context of several extensions of the Evolutionary Turing Machine model.

REFERENCES

Bonabeau, E.M., Dorigo, M., Theraulaz G., 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Burgin, M., 2005. *Superrecursive Algorithms*. Springer, New York.

Eberbach, E., 2005. Toward a theory of evolutionary computation. BioSystems 82, 1-19.

Fogel, D.B., 1995. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. IEEE Press.

Kennedy, J., Eberhart, R., 1995. Particle Swarm Optimization. In *Proc. of the 1995 IEEE Int. Conf. on Neral Networks*, 1942-1948.

Koza, J., 1992. *Genetic Programming I, II, III*. MIT Press, 1992, 1994, 1999.

Michalewicz, Z., 1996. Genetic Algorithms + Data Structures = Evolution Programs. Third edition, Springer-Verlag.

Michalewicz, Z., Fogel, D.B., 2004. *How to Solve It: Modern Heuristics*. 2nd edition, Springer-Verlag.

Russell, S., Norvig, P., 1995. Artificial Intelligence: A Modern Approach. Prentice-Hall (2nd ed. 2003).