Solution and Fitness Evolution (SAFE): Coevolving Solutions and Their Objective Functions

Moshe Sipper[1,2], Jason H. Moore[1], and Ryan J. Urbanowicz[1] ⋆

[1] Institute for Biomedical Informatics, University of Pennsylvania, Philadelphia, PA 19104-6021, USA

[2] Department of Computer Science, Ben-Gurion University, Beer Sheva 84105, Israel

{sipper,jhmoore,ryanurb}@upenn.edu

Abstract. We recently highlighted a fundamental problem recognized to confound algorithmic optimization, namely, *conflating* the objective with the objective function. Even when the former is well defined, the latter may not be obvious, e.g., in learning a strategy to navigate a maze to find a goal (objective), an effective objective function to *evaluate* strategies may not be a simple function of the distance to the objective. We proposed to automate the means by which a good objective function may be discovered—a proposal reified herein. We present **S**olution **A**nd **F**itness **E**volution (**SAFE**), a *commensalistic* coevolutionary algorithm that maintains two coevolving populations: a population of candidate solutions and a population of candidate objective functions. As proof of principle of this concept, we show that SAFE successfully evolves not only solutions within a robotic maze domain, but also the objective functions needed to measure solution quality during evolution.

Keywords: Evolutionary Computation · Coevolution · Novelty search · Objective function.

1 Objective $\neq$ Objective Function

The goal of any evolutionary algorithm (EA) is to solve a problem, i.e., obtain a specified *objective*. This invariably entails defining an *objective function* (e.g., fitness function), which is the function we want to minimize or maximize. In a recent paper, we targeted a fundamental problem that one might face in practice. Specifically, while the objective may be known and well defined, the objective function may be deceptive, and one might easily conflate the objective with the objective function [1]. Consider the mazes in Figure 1, wherein the challenge is to evolve a robotic controller (i.e., a model that determines movement given the robot's current state) such that the robot, when placed in the start position, is

⋆ © Springer Nature Switzerland AG 2019. L. Sekanina et al. (Eds.): EuroGP 2019, LNCS 11451, pp. 116, 2019. The final authenticated version is available online at https://doi.org/10.1007/978-3-030-16670-0_10.

able to make its way to the goal. The controller defines the *behavior* (i.e., decides the direction of movement) of the robot when it is in a given *state* (position in the maze, distances to obstacles). The set of movement decisions over a fixed number of time steps defines the robot's *path*, and the robot's *endpoint* is its position at the final time step.

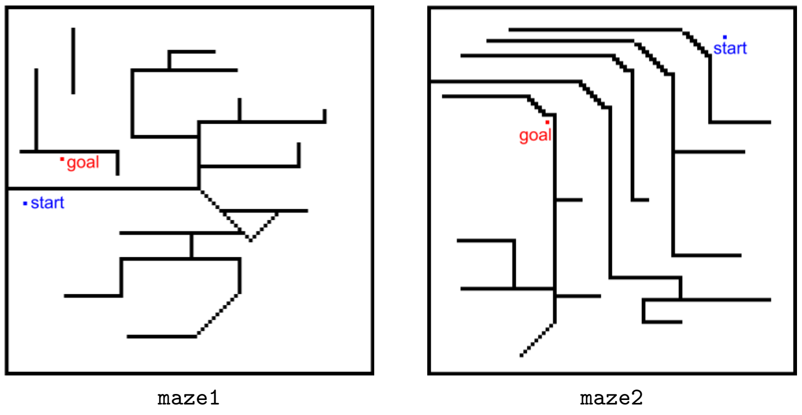

Fig. 1. In these maze problems a robot begins at the start square and must make its way to the goal square (objective).

It seems intuitive that the fitness f of a given robotic controller be defined as a function of the distance from the robot to the objective, as done, e.g., by [2]. However, reaching the objective may be difficult since the robot is faced with a deceptive landscape, where higher fitness (i.e., being reasonably close to the goal) may not imply that the robot is "almost there". It is quite easy for the robot to attain a fairly good fitness value, yet be stuck behind a wall in a local optimum—quite far from the objective in terms of the path needed to be taken. Indeed, our experiments with such a fitness-based evolutionary algorithm (Section 7) produced the expected failure, demonstrated in Figure 2.

Reaching the global optimum implies the acceptance of *reduced* fitness over the course of searching for an optimal controller. This is at odds with the workings of an evolutionary algorithm, which in practice is driven to optimize the objective function rather than find the best way to reach the objective. Another way to frame this problem is that while an optimal solution may be representable by the controller, it may not be learnable given this simple objective function [3]. For EAs, learnability translates to *evolvability* [4].

One solution to this conflation problem was offered by [2] in the form of novelty search, which *ignores* the objective and searches for novelty (using a novelty metric that requires careful consideration, e.g., robot endpoint novelty). However, novelty for the sake of novelty alone lacks incentive for solutions that reach and stay at the objective. In [1] we offered an alternative view, namely, that the problem may lie with our ignorance of the *correct* objective function.

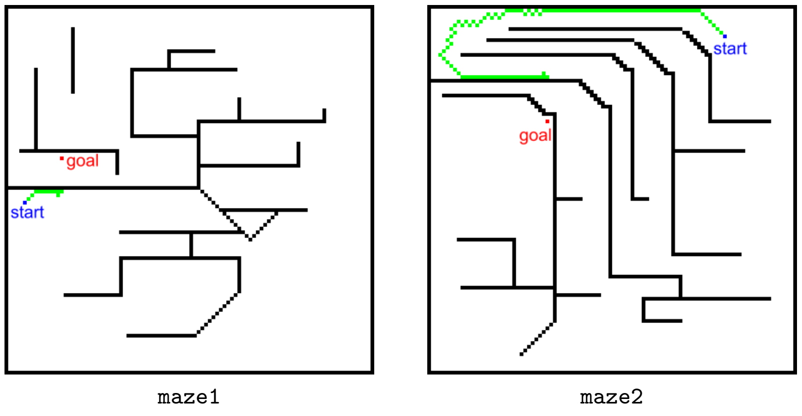

Fig. 2. Path (green) of a robot evolved by a standard evolutionary algorithm with fitness measured as distance-to-goal, evidencing how conflating the objective with the objective function leads to a non-optimal solution.

We proposed that rather than exclusively seek out novelty or rely on a fixed objective-based metric to evaluate both the "journey" and the "destination" it might be more effective to separate the optimization of the solution from the optimization of the objective function. For example, just because the robot's objective is to reach the goal does not necessarily imply that the objective function is distance-to-goal.

Thus, we reasoned, we are now left with the task of finding this better objective function, either manually or—perhaps more intriguingly—through some automated means; after all, if we are searching for a good objective function, why not employ a search algorithm? We concluded in [1] with a final speculation, namely, that coevolution [5,6] might offer the key to simultaneously explore solution optimality and objective-function optimality.

In this paper we flesh out our previous speculations and offer preliminary evidence that solutions can be coevolved along with the objective functions that evaluate them. Our newly proposed algorithm is dubbed **SAFE**, for **S**olution **A**nd **F**itness **E**volution.

This paper describes an exploratory study of SAFE. We show that where a standard, fitness-based evolutionary algorithm fails, SAFE succeeds, on a par with (or slighly better) than novelty search.

We examine four algorithms in this work—(1) a standard genetic algorithm, (2) novelty search, (3) SAFE, and (4) random search—for their ability to solve maze problems. After recounting related work in Section 2, we present two key ideas, novelty search (Section 3) and coevolution (Section 4). We continue with a delineation of the robot simulator (Section 5). We then present SAFE (Section 6) followed by results involving maze-solving robots (Section 7). We end with a discussion and concluding remarks (Section 8).

2 Related Work

We believe that the idea presented herein—treating the objective function as an evolving entity—has not been previously examined as such. There are a number of related, but distinct, research directions summarized below.

Fitness approximation explores the ways one can estimate the fitness function by constructing an approximate model. This can be quite useful when an explicit fitness function does not exist, or the evaluation of the fitness is computationally expensive. An excellent review on the topic by [7] divides fitness-approximation approaches into three categories: 1) *Problem approximation* tries to replace the original statement of the problem by one which is approximately the same but easier to solve (e.g., evaluate a turbine blade using computational fluid dynamics—CFD—simulations rather than in a wind tunnel); 2) *functional approximation* constructs an alternate expression for the objective function, e.g., instead of a CFD simulation of a blade define an explicit mathematical model (see also work on surrogate fitness functions [8, 9]); 3) *evolutionary approximation*, including fitness inheritance—wherein offspring fitness is estimated from parent fitness, and fitness imitation—where individuals are clustered into groups and only a group representative is evaluated whereupon it serves as a basis for estimating the fitness of the the others in the cluster. Interestingly, coevolution has been used in fitness approximation [10].

The objective function conflation problem is also distinct from the topic of *dynamic fitness*, wherein the problem itself (along with the global optimum and/or the fitness landscape) changes during run-time [11]. The prediction of weather patterns is in line with this challenge. Further, *multi-objective optimization*, where there is more than one objective to optimize at a time, is also distinct [12] given that it is typically up to the user to define a multi-objective fitness function. Multi-objective Pareto front methods are perhaps closest to the concept we propose in that they seek not to make fixed assumptions about what makes a 'good' solution/objective [12]. However, a Pareto front does not separate the objective from the objective function, and ultimately leaves the challenge of picking the 'optimal' solution up to the user, traditionally from a set of candidate 'non-dominated' solutions at the front. Ultimately, the challenges of dynamic fitness and multi-objective optimization are each vulnerable to the conflation of objective and objective function—and might benefit from a SAFE-inspired approach.

More generally, the topic of *open-ended evolution* comes to mind. This type of evolution occurs in nature, admitting no externally imposed fitness criterion, but rather an implicit, emergent, dynamical one (that could arguably be summed up as survivability) [13–15]. Indeed, the original novelty-search paper began with: "This paper establishes a link between the challenge of solving highly ambitious problems in machine learning and the goal of reproducing the dynamics of open-ended evolution in artificial life." [2] While open-ended evolution is a popular topic in the field of artificial life, we prefer to keep the focus herein on computational problem solving.

3 Novelty Search

Novelty search was introduced by [2] and applied to the examination of maze problems where robot controllers were sought as candidate solutions. Recall that a controller defines the behavior of a robot, yielding a path through the maze concluding in an endpoint position. The key idea was that instead of rewarding endpoint closeness to objective, individual solutions would be considered valuable if their endpoint diverged from prior solution endpoints (i.e., the system was driven to identify solutions that led the robot to new parts of the maze, regardless of objective closeness).

The novelty metric employed should be carefully defined based on the problem at hand. In [2], the novelty metric was defined as the Euclidean distance between the endpoints of two individuals. Note that this is essentially a *phenotypic*, or outcome-based, novelty measure. Differently, a *genotypic* novelty metric would examine differences between the architecture of the solutions themselves (e.g., the robot controllers). We will revisit this in Section 6, where we present a genotypic novelty metric for SAFE objective functions.

In later work, novelty was combined with other objectives (e.g., performance). For example, in [16], "an existing creature evolution platform is extended with multi-objective search that balances drives for both novelty and performance". However, this does not involve an evolving objective function, but rather, "The suggested approach is to provide evolution with both a novelty objective ... and a local competition objective". To wit, the objectives are provided by the user, as opposed to SAFE, wherein they are evolved. Appendix F of [17] includes a comprehensive up-to-date list of references to research involving novelty search.

Effectively, novelty search is identical to a standard evolutionary algorithm, with the fitness function replaced by the novelty metric. We seek novel behaviors rather than the objective, hoping that the former will ultimately lead to the latter. The endpoint of a candidate solution is compared to its cohorts in the current population and to an archive of past individuals whose behavioral endpoints were highly novel when they emerged. The novelty score is the average of the distances to the k nearest neighbors (k was set to 15 both by [2] and by us; see Table 1 in Section 7 for a summary of parameters).

For future problem flexibility, we implemented the novelty archive differently than [2]. There, an individual robot's endpoint entered the archive if its novelty score was above a certain threshold (which was adjusted dynamically). We chose to implement a fixed-size archive, with new points entering the archive until it fills up. Once full, a newly found point that exceeds the archive's current minimum will replace that minimum; otherwise it will not enter the archive. Thus, the archive saves the most novel points (at the time of their emergence). Our choice of a different archive implementation stems from our initial forays into other problems (e.g., function optimization) —to be reported in the future— where no threshold value was readily apparent.

To summarize, a robot (controller's) novelty score is computed by having the robot wander the maze for the allotted number of steps (300—see Table 1). Then, its endpoint is compared to all endpoints of its cohorts in the current generation *and* to all endpoints in the archive. The final score is then the average of the $k = 15$ nearest neighbors.

4 Coevolution

Coevolution refers to the simultaneous evolution of two or more species with coupled fitness [6]. Strongly related to the concept of symbiosis, coevolution can be mutualistic, parasitic, or commensalistic [18]: 1) In mutualism, different species exist in a relationship in which each individual (fitness) benefits from the activity of the other; 2) in parasitism, an organism lives on or in another organism, causing it harm; and 3) in commensalism, members of one species gain benefits while those of the other species neither benefit nor are harmed. Interestingly, though the idea of coevolution originates (at least) with Darwin—who spoke of "coadaptations of organic beings to each other" [19]—it is arguably somewhat less pervasive in the field of evolutionary computation than one might expect.

A cooperative (mutualistic) coevolutionary algorithm involves a number of independently evolving species, which come together to obtain problem solutions. The fitness of an individual depends on its ability to collaborate with individuals from other species [5, 6, 20, 21].

In a competitive (parasitic) coevolutionary algorithm the fitness of an individual is based on direct competition with individuals of other species, which in turn evolve separately in their own populations. Increased fitness of one of the species implies a reduction in the fitness of the other species [22].

We will discuss the particulars of coevolution—specifically, commensalistic coevolution—in Section 6, when we present the SAFE algorithm. To our knowledge, this is the first introduction of a commensalistic coevolutionary algorithm.

5 The Robot and the Maze

We implemented a robotic simulator, similar to the one used by [2], involving a robot moving in a two-dimensional environment that contains walls, with a controller dictating its actions, based on inputs from several sensors. The major difference from [2] is that we did not use NEAT—NeuroEvolution of Augmenting Topologies—for the controller, as we wanted full access to all code aspects, and we wanted to make our controller as interpretable as possible.

The maze environment is a two-dimensional grid of size 100×100 cells, where each cell can either be empty or (part of a) wall (refer to Figure 1). Two cells are designated as the start and goal. The robot can move horizontally or vertically, but not diagonally. The geometry is taxicab (Manhattan), i.e., the distance between two points is the sum of the absolute differences of their Cartesian coordinates.

The robot has 8 sensors (Figure 3):

– $dist_{up}$, $dist_{down}$, $dist_{left}$, $dist_{right}$: distances to the obstacle (or boundary) in the upward, downward, leftward, and rightward direction, respectively.

– $goal_{up}$, $goal_{down}$, $goal_{left}$, $goal_{right}$: indicators of whether the goal is towards the upward, downward, leftward, or rightward direction, respectively.

The *dist* values are computed by examining the robot's current position with respect to the nearest obstacle or boundary in the appropriate direction. The *goal* values are computed with respect to the robot's current position (the robot can sense the goal "beacon" through walls). If the goal is in the northeast quadrant of the robot, $goal_{up}$ and $goal_{right}$ are set to the maze height and width, respectively, while the other two *goal* sensors are set to 0. Similarly, the sensors are set when the goal is northwest, southeast, or southwest of the robot (Figure 3).

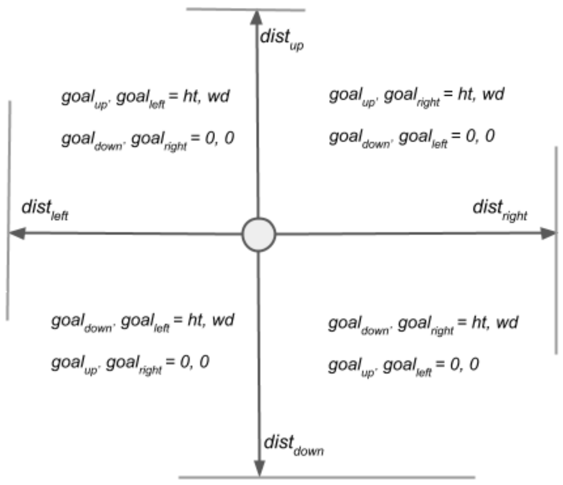
dist_up
goal_up, goal_left = ht, wd
goal_down, goal_right = 0, 0
goal_up, goal_right = ht, wd
goal_down, goal_left = 0, 0
dist_left
dist_right
goal_down, goal_left = ht, wd
goal_up, goal_right = 0, 0
goal_down, goal_right = ht, wd
goal_up, goal_left = 0, 0
dist_down

Fig. 3. The robot's 4 *dist* sensors indicate distance to nearest obstacle (wall or boundary). The 4 *goal* sensors indicate direction of goal: 2 are always set to *ht* and *wd* (maze height and width), and 2 are set to 0.

The robot moves for an allotted number of steps (set to 300, see Table 1). Movement is controlled by two variables, horizontal – h and vertical – v. The robot moves one cell in the horizontal position and one cell in the vertical position at every step, as follows:

1. Compute robot's 8 sensor values.
2. Set h and v to:

$$\begin{aligned} h = {} & p_1 dist_{right} + p_2 goal_{right} + p_3 dist_{left} + p_4 goal_{left} \\ & + p_5 dist_{down} + p_6 goal_{down} + p_7 dist_{up} + p_8 goal_{up} \end{aligned}$$

$$\begin{aligned} v = {} & p_9 dist_{right} + p_{10} goal_{right} + p_{11} dist_{left} + p_{12} goal_{left} \\ & + p_{13} dist_{down} + p_{14} goal_{down} + p_{15} dist_{up} + p_{16} goal_{up} \end{aligned}$$

3. If $h \geq 0$ ($h < 0$) then move right (left), provided adjacent cell is neither boundary nor wall (otherwise don't move).
4. If $v \geq 0$ ($v < 0$) then move down (up), provided adjacent cell is neither boundary nor wall (otherwise don't move).

The solution search space is engendered by the vector $[p_1, \dots, p_{16}] \in \mathbb{R} \cap [-1, 1]$, where each set of $[p_1, \dots, p_{16}]$ values constitutes a robot's control vector, which determines its behavior. In this domain, a successful vector will drive the robot to the goal. The algorithms we examine below all aim to find a successful control vector.

6 SAFE: Solution And Fitness Evolution

SAFE is a coevolutionary algorithm that maintains two coevolving populations: a population of candidate solutions and a population of candidate objective functions (see Figure 4). The evolution of each population is identical to a standard, single-population evolutionary algorithm—except where fitness computation is concerned. Below we describe the various components of the system: population composition, initialization, selection, elitism, crossover, mutation, and fitness computation.

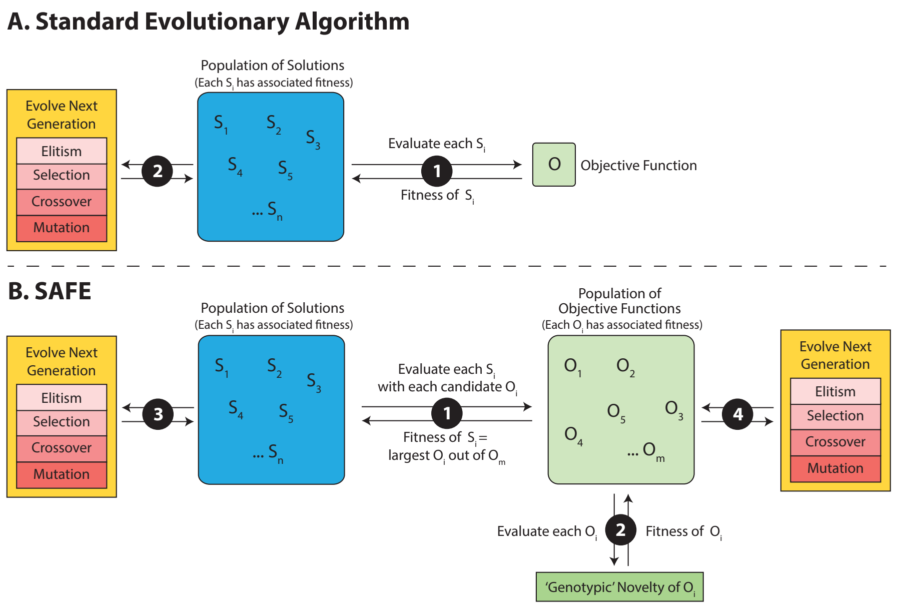
A. Standard Evolutionary Algorithm
Population of Solutions
(Each S_i has associated fitness)
Evolve Next Generation
Elitism
Selection
Crossover
Mutation
2
S_1 S_2 S_3 S_4 S_5 ... S_n
Evaluate each S_i
1
Fitness of S_i
O
Objective Function
B. SAFE
Population of Solutions
(Each S_i has associated fitness)
Population of Objective Functions
(Each O_i has associated fitness)
Evolve Next Generation
Elitism
Selection
Crossover
Mutation
3
S_1 S_2 S_3 S_4 S_5 ... S_n
Evaluate each S_i with each candidate O_i
1
Fitness of S_i = largest O_i out of O_m
O_1 O_2 O_5 O_3 O_4 ... O_m
4
Evolve Next Generation
Elitism
Selection
Crossover
Mutation
Evaluate each O_i
2
Fitness of O_i
'Genotypic' Novelty of O_i

Fig. 4. A single generation of a standard evolutionary algorithm vs. a single generation of SAFE. The numbered circles identify sequential steps in the respective algorithms.

Populations. An individual in the solutions population is a list of 16 real values, each in the range $[-1, 1]$. These values are the robot parameters, p_i, described in Section 5.

An individual in the objective-functions population is a list of 2 real values $[a, b]$, each in the range $[0, 1]$, whose usage is described below. Population sizes and other parameters are given in Table 1 (Section 7).

Initialization. For every evolutionary run: both populations are initialized to random (fixed-length) lists, wherein each component value is in the appropriate range.

Selection. Tournament selection with tournament size 5, i.e., choose 5 individuals at random from the population and return the individual with the best fitness as the selected one.

Elitism. The 2 individuals with the highest fitness in a generation are copied ("cloned") into the next generation unchanged.

Crossover. Standard single-point crossover, i.e., select a random crossover point and swap two parent genomes beyond this point to create two offspring. The crossover rate is the probability with which crossover between two selected parents occurs. (Note that for an objective-function individual, which comprises two values, if crossover occurs, it will always be, ipso facto, at the same position.)

Mutation. Mutation is done with probability 0.4 (per individual in the population) by selecting a random gene (of the 16 or 2, respectively) and replacing it with a new random value in the appropriate range.

Fitness. Fitness computation is where SAFE dynamics come into play.

In a standard evolutionary algorithm (which is one of the four algorithms we test below) each solution individual is assigned fitness based on a fixed objective function that computes a value inversely proportional to the distance to goal of the robot's endpoint, i.e., $1/\mathit{distToGoal}$ (Figure 4A).

With novelty search, an individual's fitness value is replaced with its novelty score, as described in Section 3.

In SAFE, each solution individual, S_i, $i \in \{1, \ldots, n\}$ is scored by every candidate objective-function individual, O_j, in the current population, $j \in \{1, \ldots, m\}$ (Figure 4B). In this preliminary investigation, candidate SAFE objective functions were allowed to incorporate both 'distance to goal' as well as novelty in order to calculate solution fitness. The best (highest) of these objective function scores is then assigned to the individual solution as its fitness value (Algorithm 1). Note that most of the computational cost goes into simulating the robot and computing novelty scores—which is done only once per individual solution.

As noted above, an objective-function individual is a pair $[a, b]$; specifically, a determines the influence of 'distance to goal' and b determines the influence of 'phenotypic solution novelty'. $O_j(S_i)$ is the fitness score that objective function O_j assigns to solution S_i, given as:

$$O_j(S_i) = a \times \frac{1}{\mathit{distToGoal}_i} + b \times \mathit{noveltyScore}_i\,, \tag{1}$$

Algorithm 1 Compute fitness values of solutions population

```
1: $n \leftarrow$ size of solutions population
2: $m \leftarrow$ size of objective-functions population
3: for $i \leftarrow 1$ to $n$ do
4:     simulate robot with $S_i$, and derive $endPosition_i$, $distToGoal_i$
5: for $i \leftarrow 1$ to $n$ do
6:     compute $noveltyScore_i$, based on $endPosition$ and $archive$
7: for $i \leftarrow 1$ to $n$ do
8:     for $j \leftarrow 1$ to $m$ do
9:         $f_j \leftarrow O_j(S_i)$    # $O_j$ uses $distToGoal$ & $noveltyScore$ (Equation 1)
10:    $solutionFitness_i \leftarrow \max f_j$
```

where $distToGoal_i$ is the distance to goal of robotic controller (solution) i's endpoint, and $noveltyScore_i$ is the novelty score of solution i. Rather than use one or the other, we let evolution discover an effective mix of the two objective function components (as opposed, e.g., to [23], wherein a weighted combination of fitness and novelty was used with pre-determined weights).

As for the objective-functions population, determining the quality of an evolving objective function places us in uncharted waters. Such an individual is not a solution to a problem, but rather the "guide"—or "path"—to a solution. As such, it is not clear what comprises a good measure of success.

We experimented with various forms of mutualistic coevolution, where the fitness of an objective function depends on its ability to ascribe fitness to solutions in the solutions population, such that better solutions (i.e., closer to the objective) receive higher fitness values, and worse solutions receive lower fitness values. In particular, we considered evaluating objective functions based on the correlation (i.e., Pearson's or Spearman's) between the solution fitness scores it generated and distance to the actual objective. While seemingly intuitive, this approach yielded unsatisfactory results, finding that this implementation ended up reinforcing the same local minima problem encountered when using a traditional evolutionary algorithm. Despite this failure, we plan to continue investigating the possibility of a mutualistic coevolutionary approach, where the fitness of objective functions is dependent in some way on the population of candidate solutions.

In the working version of SAFE presented here, we turned to a commensalistic coevolution strategy, where the objective functions' fitness does not depend on the population of solutions. Instead, it relies on *genotypic* novelty, based on the objective-function individual's two-valued genome, $[a, b]$. The distance between two objective functions—$[a_1, b_1]$, $[a_2, b_2]$—is simply the Euclidean distance of their genomes, given as: $\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}$. Note the contrast between genotypic novelty here and phenotypic (i.e., outcome-based) novelty used to evaluate solutions (both by novelty search and by SAFE).

Each generation, every candidate objective function is compared to its cohorts in the current population of objective functions and to an archive of past individuals whose behaviors were highly novel when they emerged. The novelty score is the average of the distances to the k ($= 15$) nearest neighbors, and is used in computing objective-function fitness (Algorithm 2).

Algorithm 2 Compute fitness values of objective-functions population

```
1: $m \leftarrow$ size of objective-functions population
2: for $i \leftarrow 1$ to $m$ do
3:     compute $noveltyScoreObj_i$
4:     $objectiveFitness_i \leftarrow noveltyScoreObj_i$
```

To clarify, we recap SAFE's double use of the idea of novelty. First, solutions evolve not via pure goal-directedness or through pure phenotypic novelty, but rather through a combination of the two. The exact nature of this combination is determined by the evolving objective functions, which themselves use novelty (albeit genotypic) to explore the objective-function search space.

The code for SAFE and all the experiments carried out in this paper is available at `https://github.com/EpistasisLab/`.

7 Results

To test our new framework, we performed eight sets of experiments, using four algorithms, each set to search for robotic controllers that solve the two mazes of Figure 1:

1. Standard evolutionary algorithm (fitness is $1/distToGoal$).
2. Novelty search.
3. SAFE.
4. Random search.

Table 1 summarizes the run parameters.

Random search serves as a yardstick to ensure evolution is tackling a non-trivial task. It is done by drawing 100,000 solutions at random and outputting the best one. 100,000 equals maximal number of generations $\times$ population size, which are used by the previous algorithms. This ensures a fair comparison in terms of resources.

The results are shown in Table 2. A solution is deemed successful if the robot gets to within a distance of 2 from the goal. We note that SAFE is on a par with novelty search on `maze1`, and able to discover slightly more solutions for the harder `maze2`. SAFE also appeared to find a solution that reached the goal within slightly fewer generations, however this finding cannot be deemed significant. Figure 5 shows sample solutions found by SAFE (contrast this with the standard evolutionary algorithm, which always gets stuck in a local minimum, as exemplified in Figure 2).

Table 1. Algorithm parameters. Unless stated, a relevant parameter not shown is that listed under 'Standard EA' (e.g., the generation count for novelty search is that of the standard EA).

Description	Value
Standard EA	
Number of evolutionary runs	500
Maximal number of generations	500
Size of solutions population	200
Type of selection	Tournament
Tournament size	5
Type of crossover	single-point
Crossover rate	0.8
Probability of mutation (solutions)	0.4
Number of top individuals copied (elitism)	2
Maximal no. steps taken by robot	300
Stop if distance to goal $\leq$	2
Novelty search	
Average over k nearest neighbors, $k =$	15
Archive size	1000
SAFE	
Size of objective-functions population	200
Probability of mutation (objective functions)	0.4
Random search	
Number of random individuals drawn	100,000

Table 2. Results of 500 evolutionary runs per each algorithm. Success: number of successful runs, where the algorithm discovered a solution that gets the robot to a distance of 2 or less from the goal. Generations: average number of generations (standard deviation) to success.

Algorithm	Maze	Success	Generations
Standard EA	`maze1`	0	—
	`maze2`	0	—
Novelty	`maze1`	322	145 (132)
	`maze2`	10	249 (163)
SAFE	`maze1`	328	141 (130)
	`maze2`	17	227 (152)
Random	`maze1`	9	—
	`maze2`	0	—

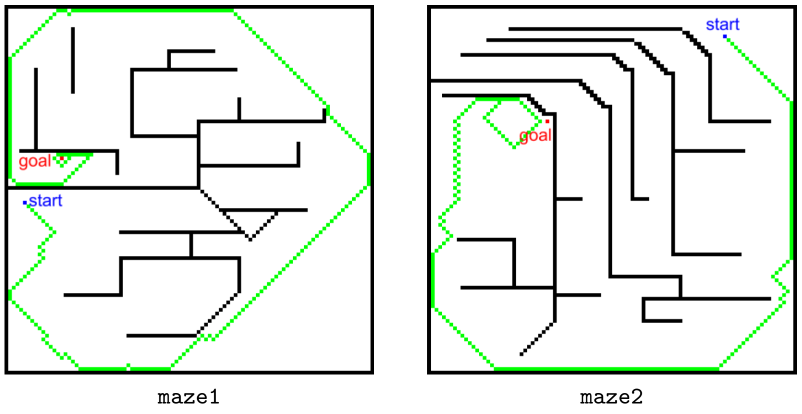

Fig. 5. Solutions to the maze problems, evolved by SAFE.

As explained in Section 6, the evolving objective function is a real-valued tuple $[a, b]$, with a being the distance-to-goal coefficient, and b being the novelty coefficient. Examining all successful objective functions evolved by SAFE for `maze1` (those that led to a maze solution), we observed that the average of a was 0.85 ($sd = 0.16$) and the average of b was 0.99 ($sd = 0.01$); for `maze2`, average a was 0.83 ($sd = 0.18$) and average b was 0.98 ($sd = 0.02$). Evolution is thus seen to strike a fairly even balance between being goal-oriented and seeking novelty, favoring the latter to a small extent. This motivated a final experiment, wherein we ran the standard evolutionary algorithm, but with the fitness function of SAFE (Equation 1) and fixed $a = b = 0.5$. The number of successful runs was 307 for `maze1` and 17 for `maze2`, on par with SAFE and novelty search. Of course, SAFE's task is more difficult, given that it is not handed the objective function but must evolve it.

Recall from Section 5 that the robot's movement is controlled by two variables, horizontal – h and vertical – v, each defined as a weighted sum of the eight sensor values: $dist_{right}$, $goal_{right}$, $dist_{left}$, $goal_{left}$, $dist_{down}$, $goal_{down}$, $dist_{up}$, $goal_{up}$. An evolved solution is a vector of 16 real values, $[p_1, \ldots, p_{16}]$, where $[p_1, \ldots, p_8]$ are the weights used by h, and $[p_9, \ldots, p_{16}]$ are the weights used by v. Given that the model is fairly white box in nature (as opposed, e.g., to a neural network's black-box essence), we might attempt to examine the evolved solutions—whose summary is shown in Table 3. For `maze1` we note that h's *left* (absolute) sensor values are quite low, perhaps because the start position is close to the left wall, so there is no point in going left. Also, the *right* values are high for both h and v, so apparently right is a good direction. For `maze2` we observe that moving left seems to be favored, and—to a lesser degree—down.

Table 3. Evolved robotic control parameters—$[p_1, \ldots, p_{16}]$—of successful solutions. Each value is the average (standard deviation) of all successful runs. Parameters are shown alongside the sensors they weight.

p	sensor	`maze1`	`maze2`
horizontal			
p_1	$dist_{right}$	0.68 (0.26)	-0.27 (0.33)
p_2	$goal_{right}$	-0.56 (0.29)	0.04 (0.38)
p_3	$dist_{left}$	0.11 (0.39)	-0.82 (0.18)
p_4	$goal_{left}$	-0.04 (0.29)	0.52 (0.4)
p_5	$dist_{down}$	-0.65 (0.25)	0.58 (0.42)
p_6	$goal_{down}$	-0.73 (0.2)	0.38 (0.45)
p_7	$dist_{up}$	0.45 (0.38)	0.11 (0.2)
p_8	$goal_{up}$	0.33 (0.28)	-0.37 (0.37)
vertical			
p_9	$dist_{right}$	0.65 (0.23)	0.1 (0.1)
p_{10}	$goal_{right}$	0.62 (0.31)	-0.09 (0.45)
p_{11}	$dist_{left}$	-0.4 (0.46)	0.78 (0.15)
p_{12}	$goal_{left}$	-0.55 (0.31)	0.54 (0.29)
p_{13}	$dist_{down}$	-0.24 (0.46)	0.11 (0.17)
p_{14}	$goal_{down}$	0.03 (0.38)	-0.15 (0.5)
p_{15}	$dist_{up}$	-0.64 (0.34)	-0.21 (0.39)
p_{16}	$goal_{up}$	0.41 (0.3)	-0.15 (0.52)

8 Discussion and Concluding Remarks

Aiming to confront the optimization conflation problem—where the objective is conflated with the objective function—we separated these two entities into two populations, and presented SAFE, a coevolutionary algorithm to evolve the two simultaneously. We showed that both the "journey" and the "destination" can be discovered together, i.e., the solution and the objective function needed to find it.

Our aim herein has been to provide preliminary proof for a novel idea. As such, we did not perform extensive experiments on parameter combinations, although recent research suggests that parameters should not matter too much [24].

Stanley [17] recently wrote that, "While a good algorithm is sometimes one that performs well, sometimes a good algorithm is instead *one that leads to other algorithms and new frontiers*." While ours is but a first foray into new territory we hope it leads to new frontiers. Indeed, there are several exploratory avenues that come to mind:

– First, we would like to add other and more complex problem domains, hoping to reinforce the promising results presented herein and identify areas where the SAFE concept is not only competitive but clearly advantageous. It is of particular interest to demonstrate whether SAFE can adapt itself to problems where little prior knowledge exists regarding either the objective or the best path to said objective.

– The coevolutionary dynamics engendered by SAFE are likely to be quite interesting and worthy of study in and of themselves. The impact of elements including elitism approach, run parameters, and convergence criteria are all worthy of future consideration.
– Our evolving objective function comprises two components, distance-to-objective and novelty. We could consider other components that might be added to the mix, e.g., the robot's distances to walls, its trajectory (behavior) through the environment [25,26], and so forth.
– We used a simple measure to drive objective-function evolution—genotypic novelty. Other, possibly better measures might be designed. As noted in Section 6 we did preliminary work on mutualistic fitness scores for objective functions, work which—though it did not pan out—we plan to continue.
– Explore the concept of whether having an objective function that changes over the course of evolutionary search may itself be an important aspect of evolvability in certain deceptive domains.

Considering the "coadaptations of organic beings to each other" [19], is it not natural to view the objective function as a constantly shifting, evolving entity? We speculate that leveraging a combination of open-endedness and coevolutionary search may offer a path to solving a variety of future problems where traditional approaches fail, characterized by deceptive landscapes, high dimensionality, and a lack of prior knowledge.

Acknowledgements

This work was supported by National Institutes of Health (USA) grants AI116794, LM010098, and LM012601.

References

1. Sipper, M., Urbanowicz, R.J., Moore, J.H.: To know the objective is not (necessarily) to know the objective function. BioData Mining 11(1) (Oct 2018)
2. Lehman, J., Stanley, K.O.: Exploiting open-endedness to solve problems through the search for novelty. In: Proceedings of the Eleventh International Conference on Artificial Life (ALIFE). MIT Press, Cambridge, MA (2008)
3. Domingos, P.: A few useful things to know about machine learning. Communications of the ACM 55(10), 78–87 (2012)
4. Wagner, G.P., Altenberg, L.: Perspective: complex adaptations and the evolution of evolvability. Evolution 50(3), 967–976 (1996)
5. Zaritsky, A., Sipper, M.: Coevolving solutions to the shortest common superstring problem. Biosystems 76(1), 209–216 (2004)
6. Pena-Reyes, C.A., Sipper, M.: Fuzzy CoCo: A cooperative-coevolutionary approach to fuzzy modeling. IEEE Transactions on Fuzzy Systems 9(5), 727–737 (2001)
7. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft computing 9(1), 3–12 (2005)

8. Buche, D., Schraudolph, N.N., Koumoutsakos, P.: Accelerating evolutionary algorithms with Gaussian process fitness function models. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 35(2), 183–194 (2005)
9. Brownlee, A.E.I., Regnier-Coudert, O., McCall, J.A.W., Massie, S.: Using a Markov network as a surrogate fitness function in a genetic algorithm. In: Proceedings of the IEEE Congress on Evolutionary Computation. pp. 1–8 (July 2010)
10. Schmidt, M.D., Lipson, H.: Coevolution of fitness predictors. IEEE Transactions on Evolutionary Computation 12(6), 736–749 (2008)
11. Grefenstette, J.J.: Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. vol. 3, pp. 2031–2038. IEEE (1999)
12. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: International Conference on Parallel Problem Solving From Nature. pp. 849–858. Springer (2000)
13. Sipper, M.: If the milieu is reasonable: Lessons from nature on creating life. Journal of Transfigural Mathematics 3(1), 7–22 (1997)
14. Sipper, M.: Machine Nature: The Coming Age of Bio-Inspired Computing. McGraw-Hill, New York (2002)
15. Banzhaf, W., Baumgaertner, B., Beslon, G., Doursat, R., Foster, J.A., McMullin, B., De Melo, V.V., Miconi, T., Spector, L., Stepney, S., et al.: Defining and simulating open-ended novelty: requirements, guidelines, and challenges. Theory in Biosciences 135(3), 131–161 (2016)
16. Lehman, J., Stanley, K.O.: Evolving a diversity of virtual creatures through novelty search and local competition. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. pp. 211–218. ACM (2011)
17. Stanley, K.O.: Art in the sciences of the artificial. Leonardo 51(2), 165–172 (2018)
18. Wikipedia: Symbiosis (2018), https://en.wikipedia.org/wiki/Symbiosis
19. Darwin, C.R.: On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. John Murray, London (1859)
20. Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation 8(1), 1–29 (2000)
21. Dick, G., Yao, X.: Model representation and cooperative coevolution for finite-state machine evolution. In: 2014 IEEE Congress on Evolutionary Computation (CEC). pp. 2700–2707. IEEE, Piscataway, NJ (2014)
22. Hillis, W.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D: Nonlinear Phenomena 42(1), 228–234 (1990)
23. Cuccu, G., Gomez, F.: When novelty is not enough. In: Di Chio, C., et al. (eds.) Applications of Evolutionary Computation. pp. 234–243. Springer, Berlin, Heidelberg (2011)
24. Sipper, M., Fu, W., Ahuja, K., Moore, J.H.: Investigating the parameter space of evolutionary algorithms. BioData Mining 11(2), 1–14 (2018)
25. Gomez, F.J.: Sustaining diversity using behavioral information distance. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. pp. 113–120. GECCO '09, ACM, New York, NY, USA (2009)
26. Doncieux, S., Mouret, J.B.: Behavioral diversity with multiple behavioral distances. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC). pp. 1427–1434. IEEE (2013)