To appear in *GECCO Surrogate-Assisted Evolutionary Optimisation Workshop* (SAEOpt), Denver, Colorado, USA, July 2016.

Adaptation of Surrogate Tasks for Bipedal Walk Optimization

Patrick MacAlpine, Elad Liebman, Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78701, USA
{patmac, eladlieb, pstone}@cs.utexas.edu

ABSTRACT

In many learning and optimization tasks, the sample cost of performing the task is prohibitively expensive or time consuming. Learning is instead often performed on a less expensive task that is believed to be a reasonable approximation or surrogate of the actual target task. This paper focuses on the challenging open problem of performing learning on an approximation of a true target task, while simultaneously adapting the surrogate task used for learning to be a better representation of the true target task. Our work is evaluated in the RoboCup 3D simulation environment where we attempt to learn configuration parameters for an omnidirectional walk engine used by humanoid soccer playing robots.

1. INTRODUCTION

In this paper we propose ideas on how to use surrogate tasks to optimize for a given task, while simultaneously learning how to adapt the surrogate tasks as we traverse different parts of the parameter space. We focus our investigation on the RoboCup 3D simulation environment in which autonomous simulated humanoid robots play soccer against each other. Optimizing values for a set of 25 parameters that control an omnidirectional walk engine has been one of the key challenges in this domain [2]. Ideally, we would want to evaluate sets of walk engine parameters directly on soccer gameplay. However, that can take days to complete. For this reason, in the past we have trained a robot how to walk directly on a hand-designed obstacle course, comprised of 11 different walking activities [3]. It has been empirically shown that doing well on the hand-designed obstacle course is correlated with gameplay success. Other approaches, such as using an obstacle course consisting of walk trajectories observed in real gameplay, have proven less successful.

While the hand-designed obstacle course has been an effective surrogate optimization task, is it possible that changing/adapting the surrogate task during learning results in a better learning rate or final walk? In this paper we begin to explore this question.

GECCO'16 Companion, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4323-7/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908961.2931712

2. WALK OPTIMIZATION TASKS

Below we describe our true optimization SoccerGameplay task, and our surrogate optimization ObstacleCourse task.

2.1 SoccerGameplay Optimization Task

The SoccerGameplay task consists of playing a five minute game of 4v4 soccer. A team is rewarded for both scoring goals and also for moving the ball toward the opponent's goal. The reward function used for this task is

$$\text{reward}_{\text{SoccerGameplay}} = (goalsFor - goalsAgainst) * \tfrac{1}{2}\text{FieldLength} + avgBallXPosition$$

where *avgBallXPosition* is the average position of the ball from the midline in the X (forward/offensive/positive and backward/defensive/negative) direction since the last goal was scored or, if neither teams scores, the average position of the ball from the beginning of the game.

When running the SoccerGameplay task we used a common fixed opponent: a baseline robot optimized with a hand-designed ObstacleCourse optimization task [2].

2.2 ObstacleCourse Optimization Task

For the ObstacleCourse task[1] the robot tries to navigate to a variety of `WAYPOINT` target positions on the field. Each target is active, one at a time for a fixed period of time, which varies from one target to the next, and the robot is rewarded based on its distance traveled toward the active target. If the robot reaches an active target, the robot receives an extra reward based on extrapolating the distance it could have traveled given the remaining time on the target. In addition to the `WAYPOINT` target positions, the robot has `STOP` targets, where it is penalized for any distance it travels. The robot is also given a penalty if it falls over.

In the following equations specifying the robot's rewards for targets, *Fall* is 5 if the robot fell and 0 otherwise, d_{target} is the distance traveled toward the target, and d_{moved} is the total distance moved. Let t_{total} be the full duration a target is active and t_{taken} be the time taken to reach the target or t_{total} if the target is not reached.

$$\text{reward}_{\texttt{WAYPOINT}} = d_{\text{target}} \frac{t_{\text{total}}}{t_{\text{taken}}} - Fall; \qquad \text{reward}_{\texttt{STOP}} = -d_{\text{moved}} - Fall$$

The duration of an ObstacleCourse task is fixed at 160 seconds. An ObstacleCourse task can be run faster than real-time, however, and thus is completed in ∼30 seconds (an order of magnitude faster than the SoccerGameplay task).

[1] ObstacleCourse optimization task video at www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2011/html/walk.html

3. ADAPTING SURROGATE WALK TASKS

Algorithm 1 Surrogate Adaptation Optimization Process

```
Input:
    N \\True task sample frequency
1: gen := 1
2: P := initializePopulationCMAES()
3: B := initializeBasisTasks()
4: loop
5:   if gen%N = 1 then
6:     trueFits := [ ]
7:     surrFits := [ ]
8:     B' := B ∪ generateNewBasisTasks(B)
9:     for each p ∈ P do
10:      trueFits := trueFits ∪ SoccerGameplay(p)
11:      for each b ∈ B' do
12:        surrFits := surrFits ∪ ObstacleCourse(b, p)
13:    B := rankBasisTasks(B', trueFits, surrFits)
14:    P := updatePopulationCMAES(trueFits)
15:  else
16:    surrFits := [ ]
17:    for each p ∈ P do
18:      surrFits := surrFits ∪ ObstacleCourse(B, p)
19:    P := updatePopulationCMAES(surrFits)
20:  gen := gen + 1
```

Pseudocode for the process of adapting surrogate tasks during optimization is shown in Algorithm 1. First, an initial set of walk parameter sets is generated by the CMA-ES algorithm [1] (line 2), and a set of randomly generated ObstacleCourse subtasks to be used as basis surrogate tasks for optimization are created (line 3).

Every Nth generation of CMA-ES the set of basis ObstacleCourse subtasks (B) is doubled in size by calling `generateNewBasisTasks()` to create B' (line 8). The `generateNewBasisTasks()` function creates new ObstacleCourse subtasks from current ones as described in Section 3.1. Each parameter set in P is then evaluated on each of the ObstacleCourse subtasks in B' (line 12) as well as on the SoccerGameplay task (line 10). Using these evaluations, the `rankBasisTasks()` function ranks all basis ObstacleCourse subtasks on how correlated they are with the SoccerGamepley task, and then sets B to be the top half of ObstacleCourse subtasks in B' with the highest correlation (line 13). How `rankBasisTasks()` computes correlations is described in Section 3.2. Finally, CMA-ES updates the population of parameter sets P for the next generation using the fitness evaluations of the parameter sets from the SoccerGameplay task (line 14).

For all non-Nth generations of CMA-ES each parameter set in P is evaluated on the ObstacleCourse task consisting of all basis ObstacleCourse subtasks in B concatenated together (line 18). These fitness evaluations are used by CMA-ES to update the population for the next generation (line 19).

3.1 ObstacleCourse Task Generation

To generate new ObstacleCourse tasks from current ones the `generateNewBasisTasks()` function uses crossover and mutation operators. To generate a crossover, two different existing ObstacleCourse subtasks are randomly chosen to be combined. Next, a random subsequence from each of these ObstacleCourse subtasks is selected, and the subsequences are concatenated to produce a new ObstacleCourse subtask.

To create a mutation variant for a given ObstacleCourse subtask each target in the ObstacleCourse subtask is with a probability of 50% randomly mutated. Mutations consist of one of four possible operations each with equal probability: removal of the target, insertion of a new target next to the existing target, toggling the target's type (`WAYPOINT` or `STOP`, and adding small random values to the target's parameters.

To expand our set of basis functions by generating K new ObstacleCourse subtasks, we create $K/2$ crossovers and $K/2$ mutations at random, and add them to the basis task set. In our experimental setting, $K = |B|$. All generated ObstacleCourse subtasks' durations are fixed to be $1/|B|$ in length by normalizing their targets' durations to sum to this value.

3.2 ObstacleCourse Task Ranking

Given the fitness of walk engine parameter sets on both the SoccerGameplay task and all basis ObstacleCourse subtasks, the `rankBasisTasks()` function ranks each of the basis ObstacleCourse subtasks on how closely correlated their fitness evaluations of walk parameter sets are to that of the SoccerGameplay task. Spearman's rank correlation, which measures the ordinal difference between two ordered sets of values, is used when ranking the ObstacleCourse subtasks.

4. RESULTS

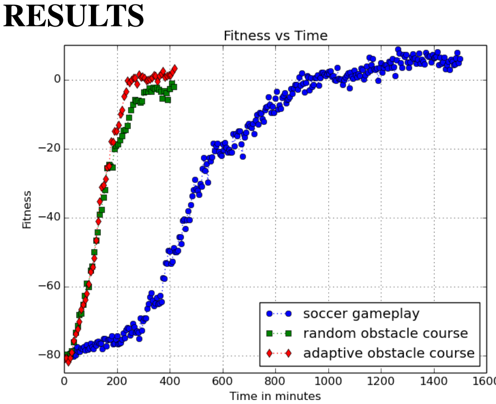

Figure 1: Learning rate on the SoccerGameplay task over time.

We optimized walk engine parameters directly on the SoccerGameplay task, on random fixed ObstacleCourse tasks, and also ObstacleCourse tasks that were adapted during optimization. For the ObstacleCourse tasks we sampled the SoccerGameplay task every fifth generation to guide learning, and when adapting ObstacleCourse tasks we used 10 basis ObstacleCourse subtasks.

Figure 1 shows learning curves over running time of the average fitnesses of CMA-ES populations on the SoccerGameplay task when using CMA-ES with a population size of 150 across 300 generations of learning. Both of the ObstacleCourse task curves were averaged across three separate optimization runs each. The results show that we are able to improve walk engine parameters when learning with ObstacleCourse surrogate optimization tasks much faster than directly using the SoccerGameplay task for optimization. Additionally, adapting the ObstacleCourse task during optimization shows some improvement over using a random fixed ObstacleCourse task, and achieves close to the same performance as directly optimizing with the SoccerGameplay task.

5. REFERENCES

[1] N. Hansen. *The CMA Evolution Strategy: A Tutorial*, January 2009. http://www.lri.fr/~hansen/cmatutorial.pdf.

[2] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, July 2012.

[3] P. MacAlpine, E. Liebman, and P. Stone. Simultaneous learning and reshaping of an approximated optimization task. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2013.