Evolved Parameterized Selection for Evolutionary Algorithms

Samuel Richter (snr359@mst.edu)

Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri 65409

December 13, 2018

Approved by

Daniel Tauritz, Advisor
Patrick Taylor
Samuel Mulder

Introduction

- Evolutionary Algorithms (EAs) are applied to a wide variety of problems, such as optimization, modeling, and system design

Introduction

- Evolutionary Algorithms (EAs) are applied to a wide variety of problems, such as optimization, modeling, and system design
- Unlike human-designed solutions, they make *a priori* no assumptions about where an optimal solution lies

Introduction

- Evolutionary Algorithms (EAs) are applied to a wide variety of problems, such as optimization, modeling, and system design
- Unlike human-designed solutions, they make *a priori* no assumptions about where an optimal solution lies
- EAs are highly parallelizable, and well-equipped to navigate complex fitness landscapes

Introduction

- Evolutionary Algorithms (EAs) are applied to a wide variety of problems, such as optimization, modeling, and system design
- Unlike human-designed solutions, they make *a priori* no assumptions about where an optimal solution lies
- EAs are highly parallelizable, and well-equipped to navigate complex fitness landscapes
- However, many EAs are highly sensitive to configurations/parameters

Introduction

There are several solutions to configuration sensitivity:

There are several solutions to configuration sensitivity:

- Self-adaptive parameters

There are several solutions to configuration sensitivity:

- Self-adaptive parameters
- Parameter-less EAs

Introduction

There are several solutions to configuration sensitivity:

- Self-adaptive parameters
- Parameter-less EAs
- Parameter-tuning/Hyper-heuristics

- With Hyper-heuristics, we generate new algorithms, or parts of algorithms, tuned for a specific purpose, such as improvement of EAs

Introduction

- With Hyper-heuristics, we generate new algorithms, or parts of algorithms, tuned for a specific purpose, such as improvement of EAs
- We can target components of an EA for improvement on a particular problem class

Introduction

- With Hyper-heuristics, we generate new algorithms, or parts of algorithms, tuned for a specific purpose, such as improvement of EAs
- We can target components of an EA for improvement on a particular problem class
- Previous work has shown that many EA components are good candidates for targeted improvement: mutation/mating preference/genetic representation/crossover operators

Introduction

- With Hyper-heuristics, we generate new algorithms, or parts of algorithms, tuned for a specific purpose, such as improvement of EAs
- We can target components of an EA for improvement on a particular problem class
- Previous work has shown that many EA components are good candidates for targeted improvement: mutation/mating preference/genetic representation/crossover operators
- Our work targets <u>selection</u> for improvement

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies
- Selection is used for:

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies
- Selection is used for:
  - Selecting individuals to be used for recombination

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies
- Selection is used for:
  - Selecting individuals to be used for recombination
  - Selecting individuals to survive to the next generation

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies
- Selection is used for:
  - Selecting individuals to be used for recombination
  - Selecting individuals to survive to the next generation
  - Selecting individuals to update internal variables (mean search space, evolution path, etc.)

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies
- Selection is used for:
  - Selecting individuals to be used for recombination
  - Selecting individuals to survive to the next generation
  - Selecting individuals to update internal variables (mean search space, evolution path, etc.)
- EAs and EC often use selection to control the path of evolution

Introduction

- Selection processes show up in many EAs, and other meta-heuristic strategies
- Selection is used for:
  - Selecting individuals to be used for recombination
  - Selecting individuals to survive to the next generation
  - Selecting individuals to update internal variables (mean search space, evolution path, etc.)
- EAs and EC often use selection to control the path of evolution
- Performance depends heavily on good selection methods

Introduction

- A number of conventional selection functions exist (tournament, truncation, fitness proportional) and are easy to implement

Introduction

- A number of conventional selection functions exist (tournament, truncation, fitness proportional) and are easy to implement
- Some selection functions are parameterized, adding more parameters to be set or tuned

Introduction

- A number of conventional selection functions exist (tournament, truncation, fitness proportional) and are easy to implement
- Some selection functions are parameterized, adding more parameters to be set or tuned
- Using the right selection functions and selection parameters (where applicable) has a large impact on EA performance

Introduction

- Even state-of-the-art EAs have insufficient performance on very difficult problems

Introduction

- Even state-of-the-art EAs have insufficient performance on very difficult problems
- The "No Free Lunch Theorem" implies that any possible selection function is optimal for some particular EA applied to some particular problem

Introduction

- Even state-of-the-art EAs have insufficient performance on very difficult problems
- The "No Free Lunch Theorem" implies that any possible selection function is optimal for some particular EA applied to some particular problem
- It is highly unlikely, for a given EA and problem, that the optimal selection exists within the conventional functions

Introduction

- Even state-of-the-art EAs have insufficient performance on very difficult problems
- The “No Free Lunch Theorem” implies that any possible selection function is optimal for some particular EA applied to some particular problem
- It is highly unlikely, for a given EA and problem, that the optimal selection exists within the conventional functions
- Therefore, we can expect a performance increase by developing a new selection function, tuned to the EA and problem

Introduction

- Our approach is to use a Hyper-heuristic, with both generative and perturbative components, to develop new selection functions, tuned to particular EAs and problem classes

Introduction

- Our approach is to use a Hyper-heuristic, with both generative and perturbative components, to develop new selection functions, tuned to particular EAs and problem classes
- We develop a meta-EA and a new selection function representation to explore the space of possible selection functions

Introduction

- Our approach is to use a Hyper-heuristic, with both generative and perturbative components, to develop new selection functions, tuned to particular EAs and problem classes
- We develop a meta-EA and a new selection function representation to explore the space of possible selection functions
- We determine quality of evolved selection functions by running the EA with them, keeping all other parameters (if any) constant

Introduction

- Our approach is to use a Hyper-heuristic, with both generative and perturbative components, to develop new selection functions, tuned to particular EAs and problem classes
- We develop a meta-EA and a new selection function representation to explore the space of possible selection functions
- We determine quality of evolved selection functions by running the EA with them, keeping all other parameters (if any) constant
- The EA's performance when using the evolved selection function (as opposed to a conventional selection function) is used as an indicator of the selection function's quality

Overview

- Objective: use a Hyper-Heuristic to generate a selection function for a particular EA operating on a particular problem class

Overview

- Objective: use a Hyper-Heuristic to generate a selection function for a particular EA operating on a particular problem class
- Step 1: define a representation for selection functions to form a search space

Overview

- Objective: use a Hyper-Heuristic to generate a selection function for a particular EA operating on a particular problem class
- Step 1: define a representation for selection functions to form a search space
- Step 2: discover which selection processes of the EA can be improved

Overview

- Objective: use a Hyper-Heuristic to generate a selection function for a particular EA operating on a particular problem class
- Step 1: define a representation for selection functions to form a search space
- Step 2: discover which selection processes of the EA can be improved
- Step 3: explore the space of selection functions with a meta-EA

Overview

- Objective: use a Hyper-Heuristic to generate a selection function for a particular EA operating on a particular problem class
- Step 1: define a representation for selection functions to form a search space
- Step 2: discover which selection processes of the EA can be improved
- Step 3: explore the space of selection functions with a meta-EA
- Step 4: test the EA on new instances from the same problem class to test generalization of the performance increase

Representation

- A straightforward way to represent selection functions would be to employ a Turing-complete Genetic Programming (GP) space of algorithms that takes a population as input, and returns an individual or pool of individuals

Representation

- A straightforward way to represent selection functions would be to employ a Turing-complete Genetic Programming (GP) space of algorithms that takes a population as input, and returns an individual or pool of individuals
- However, searching through the space of Turing-complete algorithms presents several difficulties

Representation

- A straightforward way to represent selection functions would be to employ a Turing-complete Genetic Programming (GP) space of algorithms that takes a population as input, and returns an individual or pool of individuals
- However, searching through the space of Turing-complete algorithms presents several difficulties
  - Searching through an infinite space of functions

Representation

- A straightforward way to represent selection functions would be to employ a Turing-complete Genetic Programming (GP) space of algorithms that takes a population as input, and returns an individual or pool of individuals
- However, searching through the space of Turing-complete algorithms presents several difficulties
  - Searching through an infinite space of functions
  - Ensuring that all outputs are valid valid selection functions

Representation

- We instead represent selection functions with a new representation, consisting of two major components

Representation

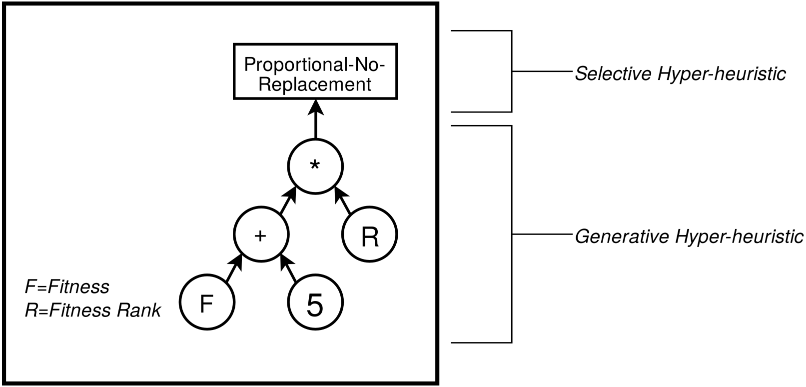
- We instead represent selection functions with a new representation, consisting of two major components
- The first component is a mathematical function, encoded in a Koza-style GP-Tree, that calculates a real-valued number corresponding to the individual's desirability

Representation

- We instead represent selection functions with a new representation, consisting of two major components
- The first component is a mathematical function, encoded in a Koza-style GP-Tree, that calculates a real-valued number corresponding to the individual's desirability
- The second component is a method of selecting individuals, based on their calculated desirability

Representation

A selection function is represented by a mathematical desirability function (encoded in a parse tree), and the final selection method.



Representation - Psuedocode

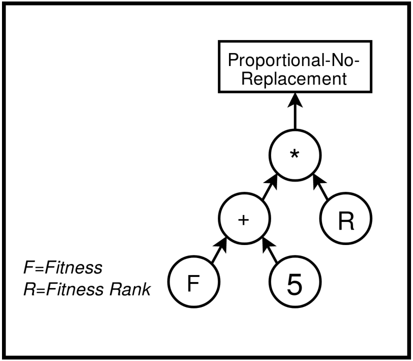
Algorithm 1 Probabilistic Selection Function

```
procedure ExampleSelection(P, W)
    W ← 0, ∀p ∈ P
    for all p ∈ P do
        W(i) ← (p.Fitness + 5) ∗ p.FitnessRank
    end for
    w_min ← minimum(W)
    s ← 0
    for all w ∈ W do
        if w_min < 0 then
            s ← s + (w − w_min)
        else
            s ← s + w
        end if
    end for
    selected ← ∅
    for j ← 1, m do
        r ← random(0, s)
        i ← 1
        while r > W(i) do
            r ← r − W(i)
            i ← i + 1
        end while
        selected ← selection ∪ P(i)
        P ← P − P(i)
    end for
end procedure
```

Representation - Psuedocode

Algorithm 1 Probabilistic Selection Function

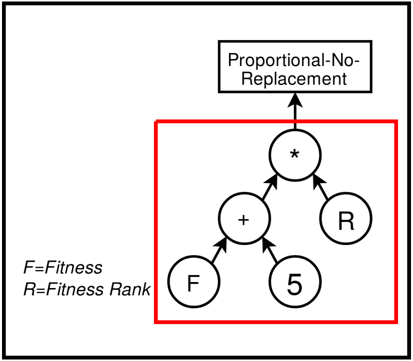
procedure EXAMPLESELECTION(P, W)
    $W \leftarrow 0, \forall p \in P$
    for all $p \in P$ **do**
        $W(i) \leftarrow (p.Fitness + 5) * p.FitnessRank$
    end for
    $w_{min} \leftarrow minimum(W)$
    $s \leftarrow 0$
    for all $w \in W$ **do**
        if $w_{min} < 0$ **then**
            $s \leftarrow s + (w - w_{min})$
        else
            $s \leftarrow s + w$
        end if
    end for
    $selected \leftarrow \emptyset$
    for $j \leftarrow 1, m$ **do**
        $r \leftarrow random(0, s)$
        $i \leftarrow 1$
        while $r > W(i)$ **do**
            $r \leftarrow r - W(i)$
            $i \leftarrow i + 1$
        end while
        $selected \leftarrow selection \cup P(i)$
        $P \leftarrow P - P(i)$
    end for
end procedure



Representation - Psuedocode

Algorithm 1 Probabilistic Selection Function

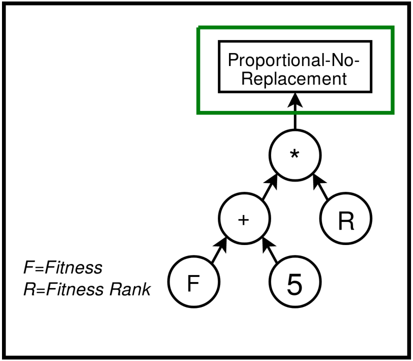
procedure ExampleSelection(P, W)
  $W \leftarrow 0, \forall p \in P$
  for all $p \in P$ **do**
    $W(i) \leftarrow \boxed{(p.Fitness + 5) * p.FitnessRank}$
  end for
  $w_{min} \leftarrow minimum(W)$
  $s \leftarrow 0$
  for all $w \in W$ **do**
    if $w_{min} < 0$ **then**
      $s \leftarrow s + (w - w_{min})$
    else
      $s \leftarrow s + w$
    end if
  end for
  $selected \leftarrow \emptyset$
  for $j \leftarrow 1, m$ **do**
    $r \leftarrow random(0, s)$
    $i \leftarrow 1$
    while $r > W(i)$ **do**
      $r \leftarrow r - W(i)$
      $i \leftarrow i + 1$
    end while
    $selected \leftarrow selection \cup P(i)$
    $P \leftarrow P - P(i)$
  end for
end procedure



Representation - Psuedocode

Algorithm 1 Probabilistic Selection Function

```
procedure ExampleSelection(P, W)
    W ← 0, ∀p ∈ P
    for all p ∈ P do
        W(i) ← (p.Fitness + 5) * p.FitnessRank
    end for
    w_min ← minimum(W)
    s ← 0
    for all w ∈ W do
        if w_min < 0 then
            s ← s + (w − w_min)
        else
            s ← s + w
        end if
    end for
    selected ← ∅
    for j ← 1, m do
        r ← random(0, s)
        i ← 1
        while r > W(i) do
            r ← r − W(i)
            i ← i + 1
        end while
        selected ← selection ∪ P(i)
        P ← P − P(i)
    end for
end procedure
```



- The components are evolved together in one genome to form one selection function, and selection functions can be evolved together to tune multiple selection steps of an EA (e.g., parent and survival selection) together

- The components are evolved together in one genome to form one selection function, and selection functions can be evolved together to tune multiple selection steps of an EA (e.g., parent and survival selection) together
- The GP-Tree is evolved with a generative Hyper-heuristic, exploring a large, yet restricted, space of mathematical functions

- The components are evolved together in one genome to form one selection function, and selection functions can be evolved together to tune multiple selection steps of an EA (e.g., parent and survival selection) together
- The GP-Tree is evolved with a generative Hyper-heuristic, exploring a large, yet restricted, space of mathematical functions
- The selection method is evolved with a perturbative Hyper-heuristic, picking from a list of methods inspired by conventional selection functions

Representation

- This format allows us a more manageable search space of selection functions, while also being robust enough to represent both conventional selection functions and a wide variety of new selection functions

Representation

- This format allows us a more manageable search space of selection functions, while also being robust enough to represent both conventional selection functions and a wide variety of new selection functions
- In addition, we can guarantee that all functions in this space are valid selection functions

Representation

- This format allows us a more manageable search space of selection functions, while also being robust enough to represent both conventional selection functions and a wide variety of new selection functions
- In addition, we can guarantee that all functions in this space are valid selection functions
- We cannot guarantee that all possible selection functions can be represented in this format, but this is an acceptable trade-off for the more easily-searchable space of selection functions

Representation - Terminals

Terminal	Description
Fitness	Individual's fitness value
Fitness Rank	Rank assigned to the individual when the population is sorted by fitness (higher fitness = higher rank)
Relative Fitness	Individual's fitness value, minus the minimum fitness value in the population, divided by the range of the population fitness values
Birth Generation	Generation number that the individual first appeared in the population

Representation - Terminals

Terminal	Description
Relative Uniqueness	Cartesian distance between the individual's genome and the centroid of all genomes in the population.
Population Size	Number of individuals in the population
Min Fitness	Smallest fitness value in the population
Max Fitness	Largest fitness value in the population
Sum Fitness	Sum of all fitness values in the population
Generation Number	Number of generations of individuals that have been evaluated since the beginning of evolution

Representation - Terminals

Terminal	Description
Constant	Returns a constant number, which is generated from a uniform selection within a configured range when the selection function is generated and held constant for the entire lifetime of the selection function.
Random	Returns a random number, which is generated from a uniform selection within a configured range every time selection is performed.

Representation - Operators

Operator	Operands	Description
$+$	2	Adds the left and right operands.
-	2	Subtracts the right operand from the left operand.
*	2	Multiplies the left and right operands.
/	2	Divides the left operand by the right operand. If the right operand is 0, the left operand is instead divided by a very small number, returning a large number while preserving the sign of the dividend.
Min	2	Returns the minimum of the left and right operands.

Representation - Operators

Operator	Operands	Description
Max	2	Returns the maximum of the left and right operands.
Step	2	Returns 1 if the left operand is greater than or equal to the right operand, and 0 otherwise.
Absolute Value	1	Returns the absolute value of the operand.

Representation - Selection Methods

Method	**Description**
Proportional-Replacement	A weighted random selection, with each individual's weight equal to its desirability score
Proportional-No-Replacement	As with Proportional-Replacement, but an individual is removed from the selection pool after being selected
k-Tournament-Replacement	A random subset of k individuals is considered, and the individual with the highest desirability score in the subset is selected
k-Tournament-No-Replacement	As with k-Tournament-Replacement, but an individual is removed from the selection pool after being selected

Method	**Description**
Truncation	Individuals with the highest desirability score are selected, with no individual being selected more than once
Stochastic-Universal-Sampling	A variant of proportional selection that chooses individuals at evenly spaced intervals, reducing sampling bias

Representation - Example

Selection Function: $(Fitness + 5) * FitnessRank \rightarrow$ Proportional-No-Replacement

Population Member	1	2	3	4
Fitness	300	250	200	350
Fitness Rank	3	2	1	4

Representation - Example

Selection Function: $(\mathit{Fitness} + 5) * \mathit{FitnessRank} \rightarrow$ Proportional-No-Replacement

Population Member	1	2	3	4
Fitness	300	250	200	350
Fitness Rank	3	2	1	4
$\mathit{Fitness} + 5$	305	255	205	355

Representation - Example

Selection Function: $(\mathit{Fitness} + 5) * \mathit{FitnessRank} \rightarrow$ Proportional-No-Replacement

Population Member	1	2	3	4
Fitness	300	250	200	350
Fitness Rank	3	2	1	4
$\mathit{Fitness} + 5$	305	255	205	355
$(\mathit{Fitness} + 5) * \mathit{FitnessRank}$	915	510	205	1420

Representation - Example

Selection Function: $(Fitness + 5) * FitnessRank \rightarrow$ Proportional-No-Replacement

Population Member	1	2	3	4
Fitness	300	250	200	350
Fitness Rank	3	2	1	4
$Fitness + 5$	305	255	205	355
$(Fitness + 5) * FitnessRank$	915	510	205	1420

Desirability scores: 915, 510, 205, 1420. Individuals are selected with probability proportional to their desirability score.

Representation - Example

Selection Function: $FitnessRanking * 100/(100 - Fitness) \rightarrow$ k-tournament, k=3

Population Member	1	2	3	4
Fitness	300	250	200	350
Fitness Rank	3	2	1	4
$FitnessRanking * 100/(100 - Fitness)$	-1.5	-1.33	-1.0	-1.6

Desirability scores: -1.5, -1.33, -1.0, -1.6. Selection picks a random set of three individuals, and selects the individual among them with the highest desirability.

Meta-EA

- We use Koza-style GP to evolve the trees representing the selection functions

Meta-EA

- We use Koza-style GP to evolve the trees representing the selection functions
- Each function is evaluated by running an EA utilizing that function on a fixed set of benchmark instances from a selected problem class

Meta-EA

- We use Koza-style GP to evolve the trees representing the selection functions
- Each function is evaluated by running an EA utilizing that function on a fixed set of benchmark instances from a selected problem class
- The best fitness reached by the EA, averaged over all runs and across all problem instances, is assigned as the fitness of the selection function at the meta-EA level

Meta-EA

- We use Koza-style GP to evolve the trees representing the selection functions
- Each function is evaluated by running an EA utilizing that function on a fixed set of benchmark instances from a selected problem class
- The best fitness reached by the EA, averaged over all runs and across all problem instances, is assigned as the fitness of the selection function at the meta-EA level
- After the meta-EA is run, the selection strategies are tested for generalization on a separate set of “testing” instances from the same problem class

Initial Experiment

- Our initial experiment used an early version of the methodology to test our approach

Initial Experiment

- Our initial experiment used an early version of the methodology to test our approach
- We built a basic EA, with simple parent selection, recombination, survival selection, and mutation

Initial Experiment

- Our initial experiment used an early version of the methodology to test our approach
- We built a basic EA, with simple parent selection, recombination, survival selection, and mutation
- We targeted improvement of the parent selection stage of the meta-EA, keeping all other factors constant

Initial Experiment

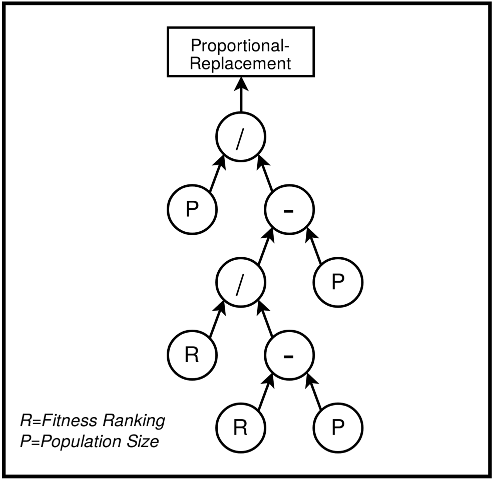
- Our initial experiment used an early version of the methodology to test our approach
- We built a basic EA, with simple parent selection, recombination, survival selection, and mutation
- We targeted improvement of the parent selection stage of the meta-EA, keeping all other factors constant
- Survival selection is performed randomly, so that the parent selection would have to provide 100% of the selection pressure

Initial Experiment

- Our initial experiment used an early version of the methodology to test our approach
- We built a basic EA, with simple parent selection, recombination, survival selection, and mutation
- We targeted improvement of the parent selection stage of the meta-EA, keeping all other factors constant
- Survival selection is performed randomly, so that the parent selection would have to provide 100% of the selection pressure
- For our benchmark problem class, we used the NK-Landscape problem class

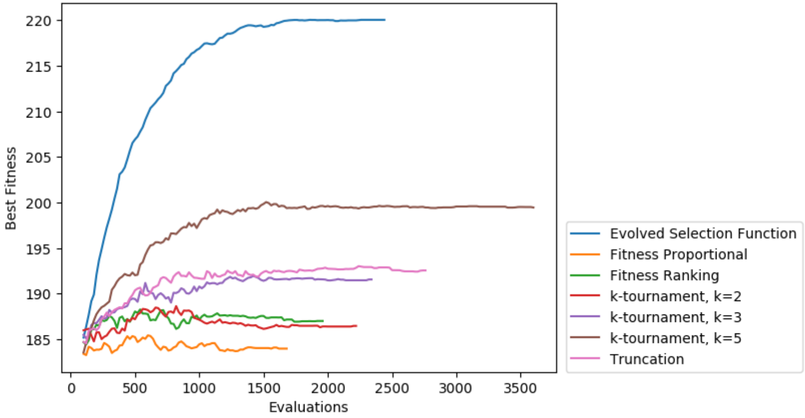
Initial Experiment - Results

Our meta-EA evolved a parent selection function that significantly outperformed typical selection on 46 of the 50 benchmark functions we tested.



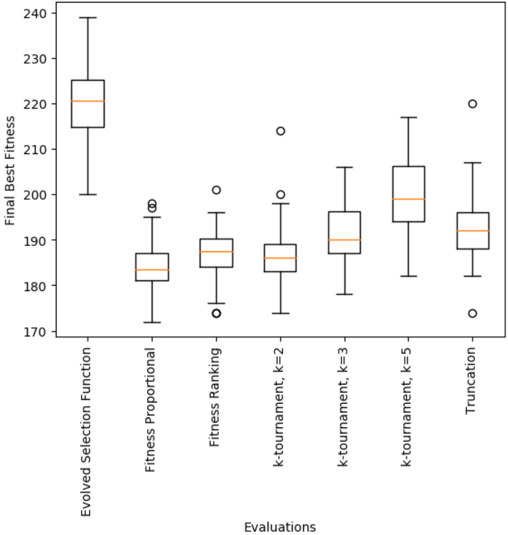
Initial Experiment - Results

Best fitness vs. EA evals, averaged over all runs:



Initial Experiment - Results

Final best fitnesses achieved by the EA:



Initial Experiment - Takeaways

Takeaways from our initial experiment:

Initial Experiment - Takeaways

Takeaways from our initial experiment:

- Our Hyper-heuristic works! We successfully evolved a new selection function for the EA, specialized to the NK problem class, that outperforms conventional selection methods. We published this work in the ECADA Workshop at GECCO 2018

Initial Experiment - Takeaways

Takeaways from our initial experiment:

- Our Hyper-heuristic works! We successfully evolved a new selection function for the EA, specialized to the NK problem class, that outperforms conventional selection methods. We published this work in the ECADA Workshop at GECCO 2018
- We next needed to test the method against a parameter-tuned EA

Initial Experiment - Takeaways

Takeaways from our initial experiment:

- Our Hyper-heuristic works! We successfully evolved a new selection function for the EA, specialized to the NK problem class, that outperforms conventional selection methods. We published this work in the ECADA Workshop at GECCO 2018
- We next needed to test the method against a parameter-tuned EA
- We also needed to show that our methodology works with other problem classes, and other EAs

Initial Experiment - Takeaways

Takeaways from our initial experiment:

- Our Hyper-heuristic works! We successfully evolved a new selection function for the EA, specialized to the NK problem class, that outperforms conventional selection methods. We published this work in the ECADA Workshop at GECCO 2018
- We next needed to test the method against a parameter-tuned EA
- We also needed to show that our methodology works with other problem classes, and other EAs
- We built these takeaways into our next three experiments

Main Experiments

We tested the most recent version of our methodology with three experiments

Main Experiments

We tested the most recent version of our methodology with three experiments

- Evolving both parent and survival selection for a basic EA, solving MK-Landscapes

Main Experiments

We tested the most recent version of our methodology with three experiments

- Evolving both parent and survival selection for a basic EA, solving MK-Landscapes
- Evolving both parent and survival selection for a basic EA, solving real-valued benchmark functions

Main Experiments

We tested the most recent version of our methodology with three experiments

- Evolving both parent and survival selection for a basic EA, solving MK-Landscapes
- Evolving both parent and survival selection for a basic EA, solving real-valued benchmark functions
- Evolving a new mean-update scheme for CMA-ES, solving real-valued benchmark functions

- We use the same basic EA as in our initial experiment, but we evolve parent and survival selection together in one genome

Experiment 1

- We use the same basic EA as in our initial experiment, but we evolve parent and survival selection together in one genome
- We use the iRace tuning package to find the best set of parameters to run the EA with, including the best choice of parent and survival selection from a list of conventional selection functions

Experiment 1

- We use the same basic EA as in our initial experiment, but we evolve parent and survival selection together in one genome
- We use the iRace tuning package to find the best set of parameters to run the EA with, including the best choice of parent and survival selection from a list of conventional selection functions
- We use the MK-Landscape problem class, a generalization of the NK-Landscape class

Experiment 1

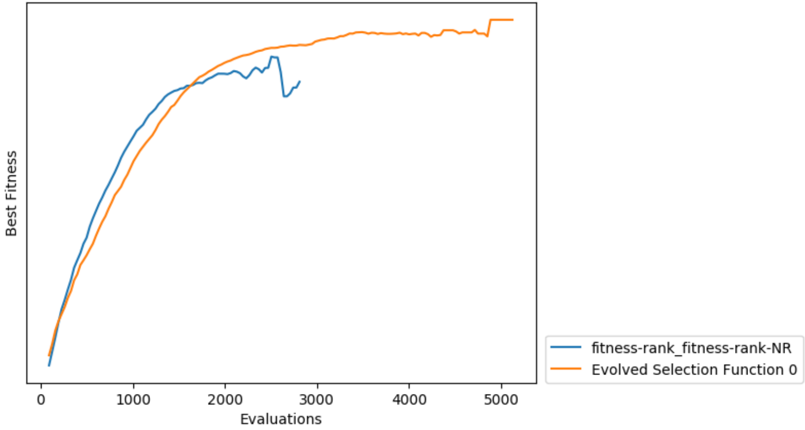
- We use the same basic EA as in our initial experiment, but we evolve parent and survival selection together in one genome
- We use the iRace tuning package to find the best set of parameters to run the EA with, including the best choice of parent and survival selection from a list of conventional selection functions
- We use the MK-Landscape problem class, a generalization of the NK-Landscape class
- At the end of the meta-EA, we test the evolved selection function against the conventional selection functions selected by iRace

Experiment 1 - Results

Results: Our meta-EA evolved a parent selection function that significantly outperformed typical selection on 45 of the 50 benchmark functions we tested.

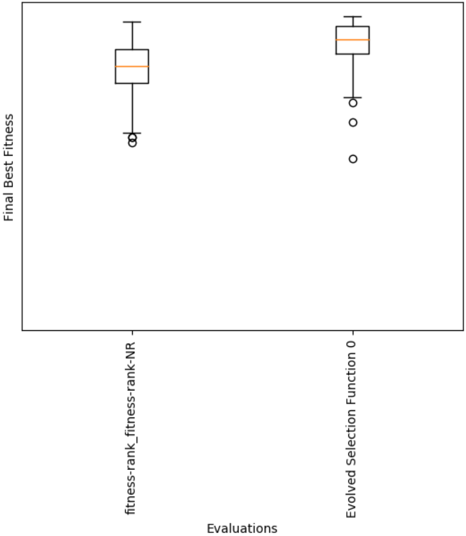
Experiment 1 - Results

Best fitness vs. EA evals, averaged over all runs:



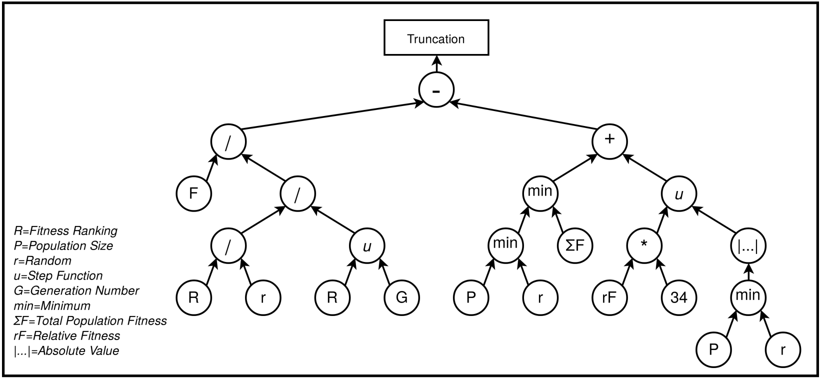
Experiment 1 - Results

Final best fitnesses achieved by the EA:



Experiment 1 - Results

Parent Selection of the best evolved selection function:



Experiment 1 - Takeaways

Experiment 1 Takeaways

- Our Hyper-heuristic performs well, even against state-of-the-art parameter tuning

Experiment 1 - Takeaways

Experiment 1 Takeaways

- Our Hyper-heuristic performs well, even against state-of-the-art parameter tuning
- Next, we need to test against more benchmark functions

Experiment 2

- We use the same basic EA as in Experiment 1, still evolving parent and survival selection together

Experiment 2

- We use the same basic EA as in Experiment 1, still evolving parent and survival selection together
- For benchmark functions, we use the 24 real-valued functions in the Comparing Continuous Optimizers package, running the meta-Ea on each function separately. We use all available the available 10-dimensional instances for each function, setting aside some for testing generalization

Experiment 2

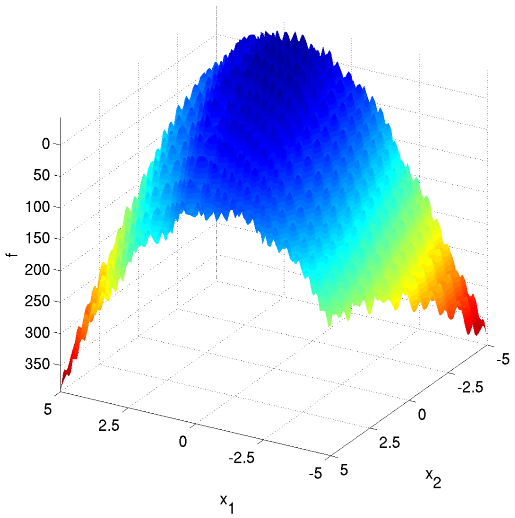
- We use the same basic EA as in Experiment 1, still evolving parent and survival selection together
- For benchmark functions, we use the 24 real-valued functions in the Comparing Continuous Optimizers package, running the meta-Ea on each function separately. We use all available the available 10-dimensional instances for each function, setting aside some for testing generalization
- We still use iRace to find the optimal parameter set for each function.

Experiment 2

- We use the same basic EA as in Experiment 1, still evolving parent and survival selection together
- For benchmark functions, we use the 24 real-valued functions in the Comparing Continuous Optimizers package, running the meta-Ea on each function separately. We use all available the available 10-dimensional instances for each function, setting aside some for testing generalization
- We still use iRace to find the optimal parameter set for each function.
- As with Experiment 1, we test for generalization at the end of the meta-EA

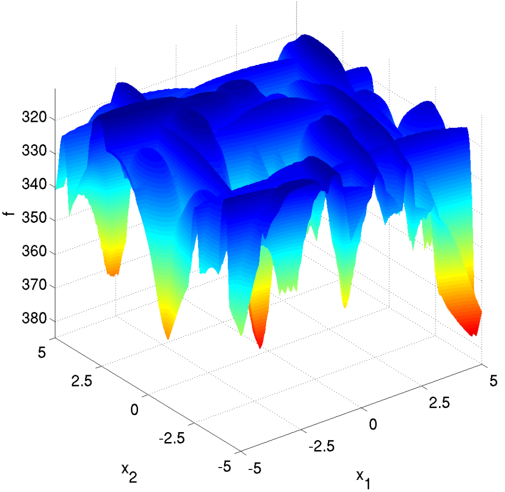
Experiment 2

COCO F #15: Rastrigin Function



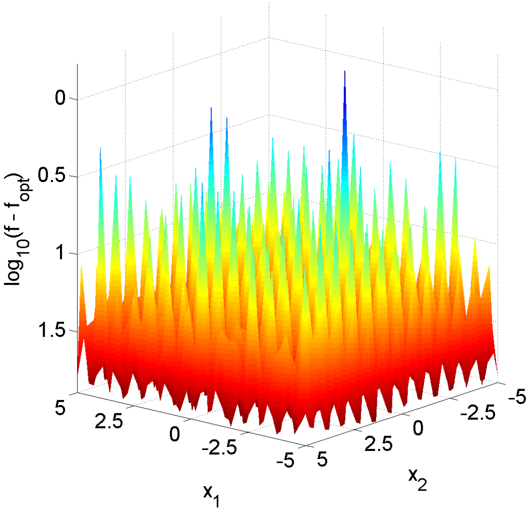
Experiment 2

COCO F #21: Gallagher's Gaussian 101-me Peaks Function



Experiment 2

COCO F #23: Katsuura Function



Experiment 2 - Results

Results:

Results:

- For 17 of the 24 function classes, the evolved selection function significantly outperformed the conventional selection on at least 1 problem instance

Experiment 2 - Results

Results:

- For 17 of the 24 function classes, the evolved selection function significantly outperformed the conventional selection on at least 1 problem instance
- For 6 of the 24 function classes, the evolved selection function significantly outperformed the conventional selection on at least half of the problem instances

Experiment 2 - Results

Results:

- For 17 of the 24 function classes, the evolved selection function significantly outperformed the conventional selection on at least 1 problem instance
- For 6 of the 24 function classes, the evolved selection function significantly outperformed the conventional selection on at least half of the problem instances
- For 2 of the 24 function classes, the evolved selection function significantly outperformed the conventional selection on all of the problem instances

Results:

Problem Index (D=10)	Number of Instances Improved
F=1	0 / 12
F=2	1 / 12
F=3	2 / 12
F=4	2 / 12
F=5	5 / 12
F=6	2 / 12
F=7	7 / 12
F=8	2 / 12
F=9	12 / 12
F=10	1 / 12
F=11	7 / 12
F=12	0 / 12

Results:

Problem Index (D=10)	Number of Instances Improved
F=13	2 / 12
F=14	0 / 12
F=15	0 / 12
F=16	0 / 12
F=17	4 / 12
F=18	4 / 12
F=19	12 / 12
F=20	0 / 12
F=21	8 / 12
F=22	6 / 12
F=23	0 / 12
F=24	1 / 12

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse
- This could be due to overspecialization to “training instances”

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse
- This could be due to overspecialization to “training instances”
- Our suspicion: some functions “too easy”; only need some kind of selection pressure

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse
- This could be due to overspecialization to “training instances”
- Our suspicion: some functions “too easy”; only need some kind of selection pressure
    - We saw little improvement on the original Rosenbrock function (F=8), but universal improvement on the rotated Rosenbrock Function.

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse
- This could be due to overspecialization to "training instances"
- Our suspicion: some functions "too easy"; only need some kind of selection pressure
  - We saw little improvement on the original Rosenbrock function (F=8), but universal improvement on the rotated Rosenbrock Function.
- Other functions "too hard"; need an entirely new evolution strategy to see significant gains

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse
- This could be due to overspecialization to "training instances"
- Our suspicion: some functions "too easy"; only need some kind of selection pressure
  - We saw little improvement on the original Rosenbrock function (F=8), but universal improvement on the rotated Rosenbrock Function.
- Other functions "too hard"; need an entirely new evolution strategy to see significant gains
  - Little improvement on functions with weak global structure, which our EA setup depends on

Experiment 2 - Takeaways

Experiment 2 Takeaways

- Unfortunately, Our Hyper-heuristic did not perform well for many of the benchmarks tested. In some cases, the evolved selection function performed worse
- This could be due to overspecialization to "training instances"
- Our suspicion: some functions "too easy"; only need some kind of selection pressure
  - We saw little improvement on the original Rosenbrock function (F=8), but universal improvement on the rotated Rosenbrock Function.
- Other functions "too hard"; need an entirely new evolution strategy to see significant gains
  - Little improvement on functions with weak global structure, which our EA setup depends on
- Using the same dimensionality for all 24 problems would inherently lead to "easy" and "hard" problems

Experiment 3

- For our third experiment, we chose a new EA: Covariance Matrix Adaptation Evolutionary Algorithm (CMA-ES)

Experiment 3

- For our third experiment, we chose a new EA: Covariance Matrix Adaptation Evolutionary Algorithm (CMA-ES)
  - Mostly parameterless ES that moves a sampling mean through the fitness space, samples points around it, updates mean

Experiment 3

- For our third experiment, we chose a new EA: Covariance Matrix Adaptation Evolutionary Algorithm (CMA-ES)
  - Mostly parameterless ES that moves a sampling mean through the fitness space, samples points around it, updates mean
  - Must more State-of-the-Art, w/ better performance than our basic EA

Experiment 3

- For our third experiment, we chose a new EA: Covariance Matrix Adaptation Evolutionary Algorithm (CMA-ES)
  - Mostly parameterless ES that moves a sampling mean through the fitness space, samples points around it, updates mean
  - Must more State-of-the-Art, w/ better performance than our basic EA
  - Able to find the global optimum of some of the COCO function classes

Experiment 3

- For our third experiment, we chose a new EA: Covariance Matrix Adaptation Evolutionary Algorithm (CMA-ES)
  - Mostly parameterless ES that moves a sampling mean through the fitness space, samples points around it, updates mean
  - Must more State-of-the-Art, w/ better performance than our basic EA
  - Able to find the global optimum of some of the COCO function classes
- We targeted the mean-update function of CMA-ES, to better select the points used for updating the mean (normally truncation)

Experiment 3

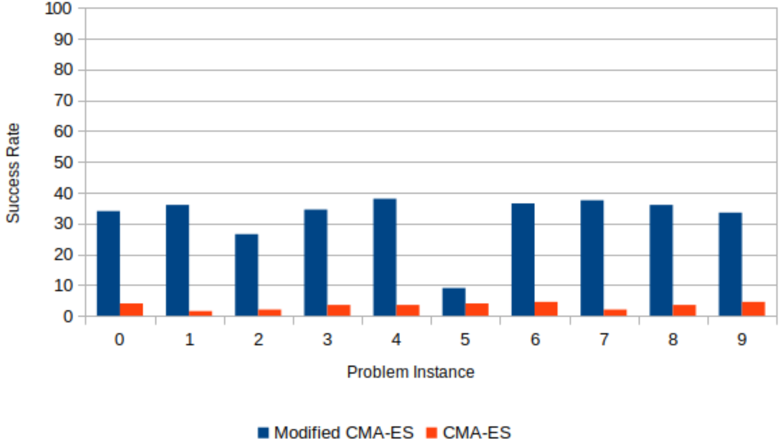
- For our third experiment, we chose a new EA: Covariance Matrix Adaptation Evolutionary Algorithm (CMA-ES)
  - Mostly parameterless ES that moves a sampling mean through the fitness space, samples points around it, updates mean
  - Must more State-of-the-Art, w/ better performance than our basic EA
  - Able to find the global optimum of some of the COCO function classes
- We targeted the mean-update function of CMA-ES, to better select the points used for updating the mean (normally truncation)
- We select the COCO functions (class and dimensionality) which CMA-ES is unable to solve more than 50% of the time

Experiment 3 - Results

Results: For 6 of the 11 functions tested, the CMA-ES using the evolved selection function found the global best fitness more often than CMA-ES without it

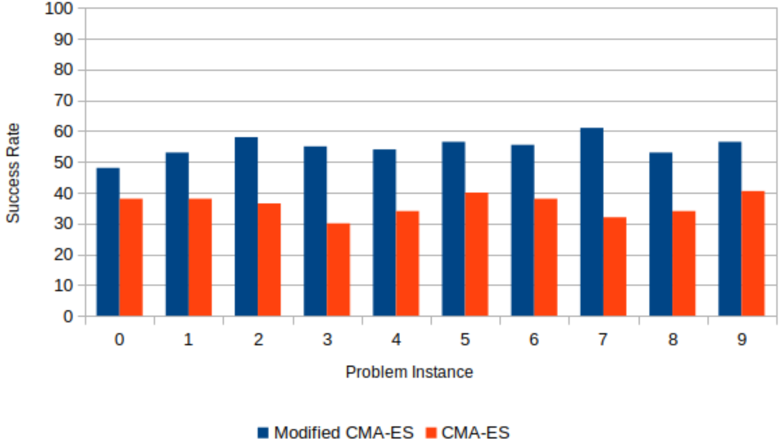
Experiment 3 - Results

F=4,D=2:



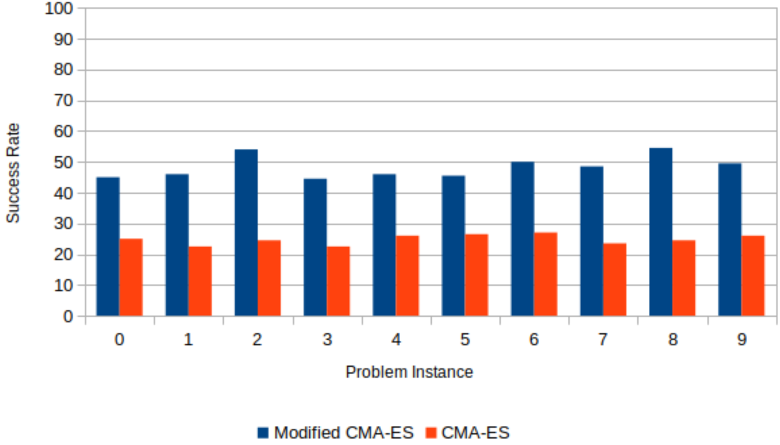
Experiment 3 - Results

F=19,D=2:



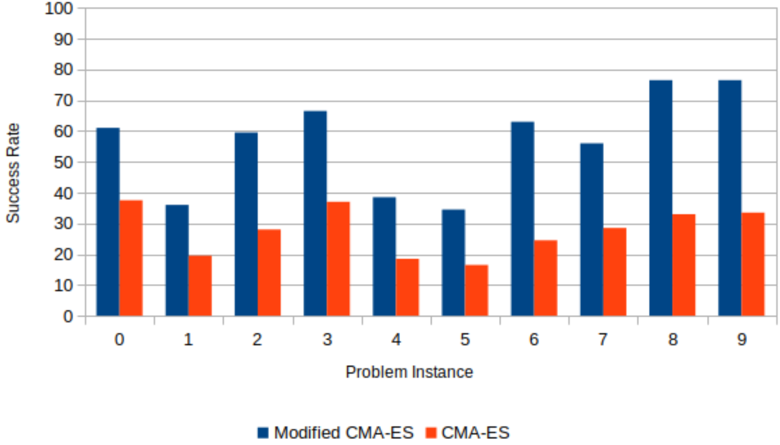
Experiment 3 - Results

F=20,D=2:



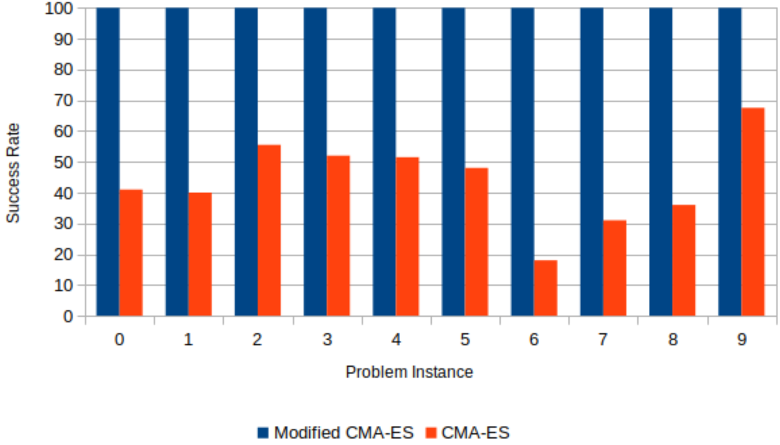
Experiment 3 - Results

F=21,D=2:



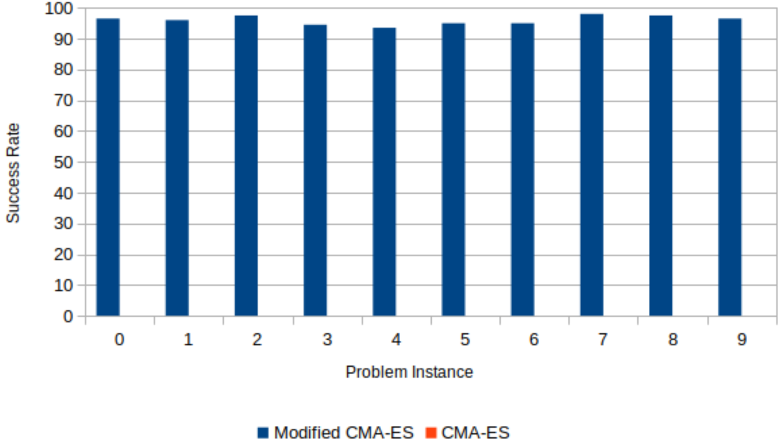
Experiment 3 - Results

F=12,D=10:



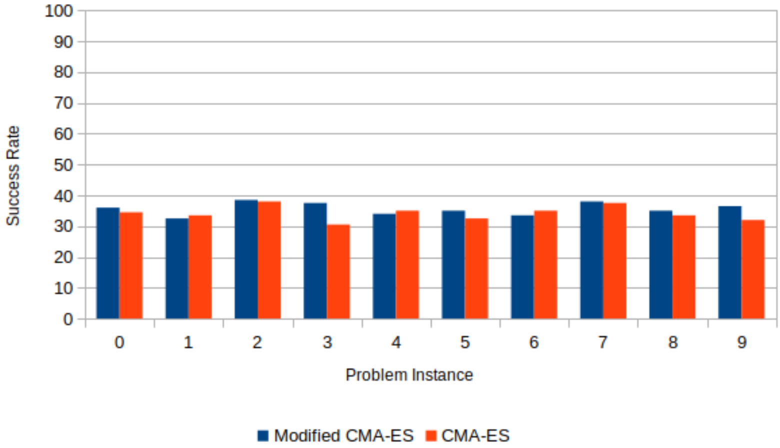
Experiment 3 - Results

F=6,D=10:



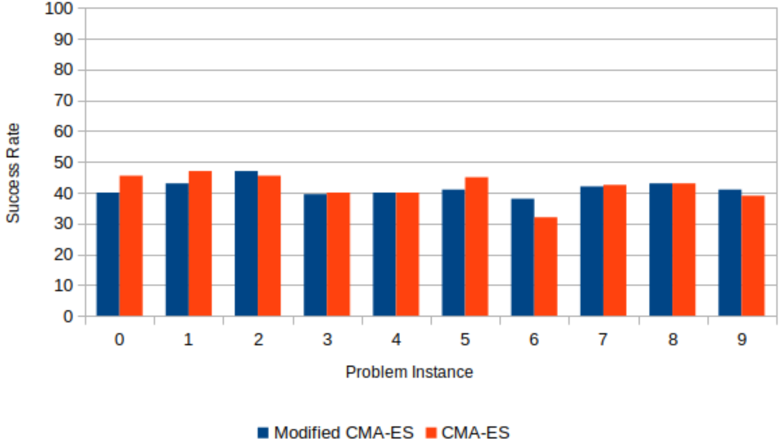
Experiment 3 - Results

F=3,D=2:



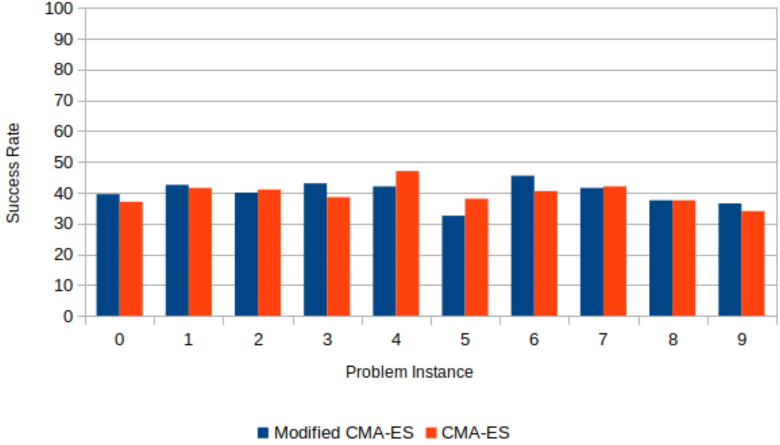
Experiment 3 - Results

F=15,D=2:



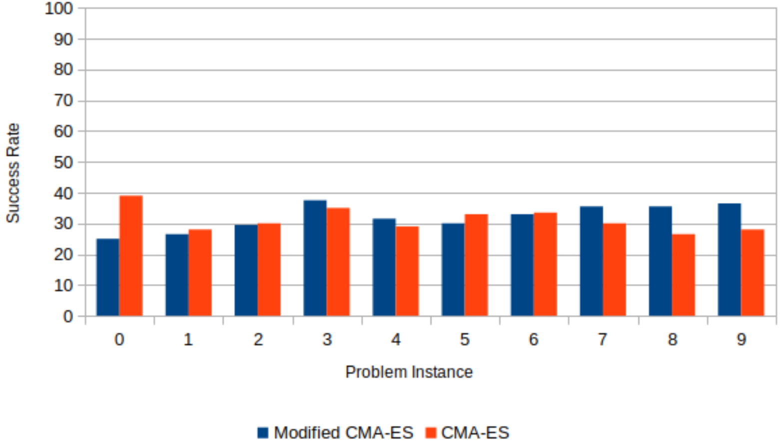
Experiment 3 - Results

F=16,D=10:



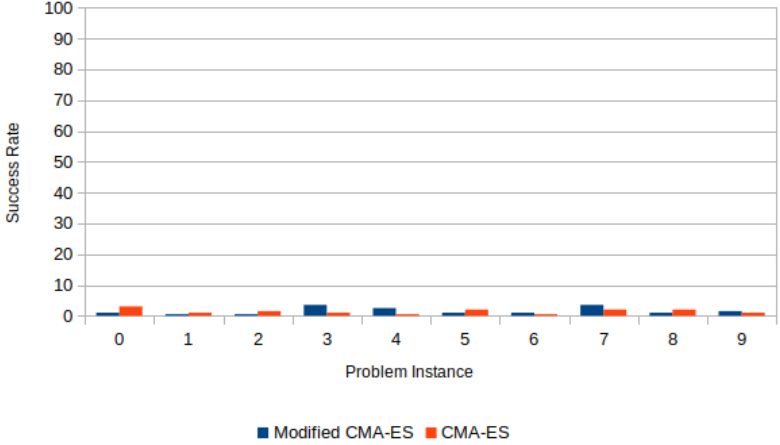
Experiment 3 - Results

F=23,D=5:



Experiment 3 - Results

F=24,D=2:



Experiment 3 Takeaways

- CMA-ES sees significant–sometimes dramatic–improvement with a new evolved mean update selection function, for some benchmark functions

Experiment 3 Takeaways

- CMA-ES sees significant–sometimes dramatic–improvement with a new evolved mean update selection function, for some benchmark functions
- Performance on other benchmarks may depend more heavily on other CMA-ES update functions

Experiment 3 - Takeaways

Experiment 3 Takeaways

- CMA-ES sees significant–sometimes dramatic–improvement with a new evolved mean update selection function, for some benchmark functions
- Performance on other benchmarks may depend more heavily on other CMA-ES update functions
- Functions with poor performance have poor local structure, require a more intelligence search of global scale (i.e., Rastrigin)

Final Conclusions

Final Conclusions

- In some cases, there is significant benefit to evolving a new selection function for a particular EA running on a particular benchmark

Final Conclusions

Final Conclusions

- In some cases, there is significant benefit to evolving a new selection function for a particular EA running on a particular benchmark
- Potential performance gain likely depends strongly on the EA, the component being replaced, and the problem being solved

Threats to Validity

Threats to Validity

- We still have yet to test this method on EAs solving real-world problems

Threats to Validity

- We still have yet to test this method on EAs solving real-world problems
- Improvements shown in experiments 1 and 2 were on very basic EA, likely too simple/out of date for real world use

Threats to Validity

Threats to Validity

- We still have yet to test this method on EAs solving real-world problems
- Improvements shown in experiments 1 and 2 were on very basic EA, likely too simple/out of date for real world use
- Implementation of CMA-ES in experiment 3 lacked restarts, and some features built into more recent CMA-ES models.

Threats to Validity

Threats to Validity

- We still have yet to test this method on EAs solving real-world problems
- Improvements shown in experiments 1 and 2 were on very basic EA, likely too simple/out of date for real world use
- Implementation of CMA-ES in experiment 3 lacked restarts, and some features built into more recent CMA-ES models.
- Some information encodable in GP-Trees is not typically considered for selection, not present in the conventional selection functions tested against.

Future Work

Future Work

- Investigate methods that do not require as much *a priori* calculation

Future Work

- Investigate methods that do not require as much *a priori* calculation
- Test methodology on EAs solving real-world problems

Future Work

Future Work

- Investigate methods that do not require as much *a priori* calculation
- Test methodology on EAs solving real-world problems
- Test against previous research on evolved selection functions (Grammatical Evolution)

Future Work

Future Work

- Investigate methods that do not require as much *a priori* calculation
- Test methodology on EAs solving real-world problems
- Test against previous research on evolved selection functions (Grammatical Evolution)
- Test on CMA-Es with more robust features/modern improvements

Future Work

Future Work

- Investigate methods that do not require as much *a priori* calculation
- Test methodology on EAs solving real-world problems
- Test against previous research on evolved selection functions (Grammatical Evolution)
- Test on CMA-Es with more robust features/modern improvements
- Test with more terminals available to GP-Tree in meta-EA, including mate preference information