Black-Box Search Algorithms

- Many complex real-world problems can be formulated as *generate-and-test* problems

- Black-Box Search Algorithms (BBSAs) iteratively generate trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge
Practitioner’s Dilemma

1. How to decide for given real-world problem whether beneficial to formulate as black-box search problem?
2. How to formulate real-world problem as black-box search problem?
3. How to select/create BBSA?
4. How to configure BBSA?
5. How to interpret result?
6. All of the above are interdependent!

Theory-Practice Gap

- While BBSAs, including evolutionary algorithms, steadily are improving in scope and performance, their impact on routine real-world problem solving remains underwhelming
- A scalable solution enabling domain-expert practitioners to routine solve real-world problems with BBSAs is needed
Tapestry Navigation

Part of the solution to bridging the gap between theory and practice is an automated approach to navigating the ever richer tapestry of BBSAs

Knight Vital on the Bayeux Tapestry

Two typical real-world problem categories

- Solving a single-instance problem: automated BBSA selection

- Repeatedly solving instances of a problem class: evolve custom BBSA
Part I: Solving Single-Instance Problems
Employing Automated BBSA Selection

Requirements

1. Need diverse set of high-performance BBSAs
2. Need automated approach to select most appropriate BBSA from set for a given problem
3. Need automated approach to configure selected BBSA
Automated BBSA Selection

1. Given a set of BBSAs, a priori evolve a set of benchmark functions which cluster the BBSAs by performance
2. Given a real-world problem, create a surrogate fitness function
3. Find the benchmark function most similar to the surrogate
4. Execute the corresponding BBSA on the real-world problem

A Priori, Once Per BBSA Set
A Priori, Once Per Problem Class

- Real-World Problem
- Sampling Mechanism
- Surrogate Objective Function
- Match with most "similar" $B_{P_k}$
- Identified most appropriate $B_{BSA_k}$

Per Problem Instance

- Real-World Problem
- Sampling Mechanism
- Surrogate Objective Function
- Apply a priori established most appropriate $B_{BSA_k}$
Requirements

1. Need diverse set of high-performance BBSAs
2. Need automated approach to select most appropriate BBSA from set for a given problem
3. Need automated approach to configure selected BBSA

Static vs. dynamic parameters

• Static parameters remain constant during evolution, dynamic parameters can change
• Dynamic parameters require parameter control
• The optimal values of the strategy parameters can change during evolution [1]
Solution

Expand tapestry with self-configuring BBSAs employing dynamic strategy parameters

Supportive Coevolution [3]
Motivation

- Algorithm and Parameter Selection
  - Problem dependent
  - Time consuming
- Dynamic Configuration
  - Optimal settings change during evolution

Self Adaptation

- Dynamical Evolution of Parameters
- Individuals Encode Parameter Values
- Indirect Fitness
- Poor Scaling
Def.: In Coevolution, the fitness of an individual is dependent on other individuals (i.e., individuals are part of the environment)

Def.: In Competitive Coevolution, the fitness of an individual is inversely proportional to the fitness of some or all other individuals

Coevolution Basics

• Primary Population
  – Encodes problem solution

• Support Population
  – Encodes configuration information
Multiple Support Populations

- Each Parameter Evolved Separately
- Propagation Rate Separated

Multiple Primary Populations
Benchmark Functions

• Rastrigin

\[ An + \sum_{i=1}^{n} \left[ x_i^2 - A \cos(2\pi x_i) \right], \forall x \in [-5.12, 5.12] \]

• Shifted Rastrigin
  – Randomly generated offset vector
  – Individuals offset before evaluation

Fitness Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>SA</th>
<th>SuCo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rastrigin N = 10</td>
<td>-0.2390 (-0.4249)</td>
<td>-0.0199 (-0.1393)</td>
</tr>
<tr>
<td>Rastrigin N = 20</td>
<td>-0.3494 (-0.4905)</td>
<td>-1.4132 (-0.7481)</td>
</tr>
<tr>
<td>Shifted N = 10</td>
<td>-4.4215 (-1.8744)</td>
<td>-2.2518 (-0.9811)</td>
</tr>
<tr>
<td>Shifted N = 20</td>
<td>-10.397 (-4.3567)</td>
<td>-5.7718 (-2.2848)</td>
</tr>
</tbody>
</table>

• All Results Statistically Significant according to T-test with \( \alpha = 0.001 \)
Conclusions

• Proof of Concept Successful
  – SuCo outperformed SA on all but one test problem

• SuCo Mutation more successful
  – Adapts better to change in population fitness

Self-Configuring Crossover [2]
Motivation

- Performance Sensitive to Crossover Selection
- Identifying & Configuring Best Traditional Crossover is Time Consuming
- Existing Operators May Be Suboptimal
- Optimal Operator May Change During Evolution

Some Possible Solutions

- Meta-EA
  - Exceptionally time consuming
- Self-Adaptive Algorithm Selection
  - Limited by algorithms it can choose from
Self-Configuring Crossover (SCX)

- Each Individual Encodes a Crossover Operator

- Crossovers Encoded as a List of Primitives
  - Swap
  - Merge

- Each Primitive has three parameters
  - Number, Random, or Inline

Applying an SCX

- Concatenate Genes

<table>
<thead>
<tr>
<th>Parent 1 Genes</th>
<th>Parent 2 Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 2.0 3.0 4.0</td>
<td>5.0 6.0 7.0 8.0</td>
</tr>
</tbody>
</table>
The Swap Primitive

- Each Primitive has a type
  - Swap represents crossovers that move genetic material

- First Two Parameters
  - Start Position
  - End Position

- Third Parameter Primitive Dependent
  - Swaps use “Width”

Applying an SCX

- Concatenate Genes
- Offspring Crossover
- Swap(3, 5, 2)
- Merge(1, r, 0.7)
- Swap(r, i, r)
The Merge Primitive

- Third Parameter Primitive Dependent
  - Merges use “Weight”

- Random Construct
  - All past primitive parameters used the Number construct
  - “r” marks a primitive using the Random Construct
  - Allows primitives to act stochastically

Applying an SCX

\[
g(i) = \alpha g(i) + (1-\alpha)g(j)
\]

\[
g(2) = 6.0 \times 0.7 + 1.0 \times 4.5
\]
The Inline Construct

- Only Usable by First Two Parameters
- Denoted as “i”
- Forces Primitive to Act on the Same Loci in Both Parents

Applying an SCX

Offspring Crossover
Swap(3, 5, 2)
Merge(1, r, 0.7)
Swap(r, i, r)
Concatenate Genes

<table>
<thead>
<tr>
<th>2.5</th>
<th>2.0</th>
<th>5.0</th>
<th>4.5</th>
<th>3.0</th>
<th>4.0</th>
<th>7.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Applying an SCX

Remove Exess Genes

Concatenate Genes

Offspring Genes

[Diagram showing Genes and Operations]

Evolving Crossovers

[Diagram showing Crossover Operations]
Empirical Quality Assessment

• Compared Against
  – Arithmetic Crossover
  – N-Point Crossover
  – Uniform Crossover

• On Problems
  – Rosenbrock
  – Rastrigin
  – Offset Rastrigin
  – NK-Landscapes
  – DTrap

<table>
<thead>
<tr>
<th>Problem</th>
<th>Comparison</th>
<th>SCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenbrock</td>
<td>-86.94 (54.54)</td>
<td>-26.47 (23.33)</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>-59.2 (6.998)</td>
<td>-0.0088 (0.021)</td>
</tr>
<tr>
<td>Offset Rastrigin</td>
<td>-0.1175 (0.116)</td>
<td>-0.03 (0.028)</td>
</tr>
<tr>
<td>NK</td>
<td>0.771 (0.011)</td>
<td>0.8016 (0.013)</td>
</tr>
<tr>
<td>DTrap</td>
<td>0.9782 (0.005)</td>
<td>0.9925 (0.021)</td>
</tr>
</tbody>
</table>

• Requires No Additional Evaluation

• Adds No Significant Increase in Run Time
  – All linear operations

• Adds Initial Crossover Length Parameter
  – Testing showed results fairly insensitive to this parameter
  – Even worst settings tested achieved better results than comparison operators

SCX Overhead
Conclusions

• Remove Need to Select Crossover Algorithm

• Better Fitness Without Significant Overhead

• Benefits From Dynamically Changing Operator

Current Work

• Combining Support Coevolution and Self-Configuring Crossover by employing the latter as a support population in the former [4]
Part II: Repeatedly Solving Problem Class Instances By Evolving Custom BBSAs Employing Meta-GP [5]

Problem

- Difficult Repeated Problems
  - Cell Tower Placement
  - Flight Routing
- Which Black-Box Search Algorithm (BBSA), such as EAs, PSOs, or SA, to use?
Possible Solutions

- Automated Algorithm Selection
- Self-Adaptive Algorithms
- Meta-Algorithms
  - Evolving Parameters / Selecting Operators
  - Evolve the Algorithm

Our Solution

- Evolving BBSAs employing meta-GP
- Post-ordered parse tree
- Evolve a repeated iteration
Our Solution

- Genetic Programing
- Post-ordered parse tree
- Evolve a repeated iteration
- High-level operations
Parse Tree

- Iteration
- Sets of Solutions
- Root returns ‘Last’ set

Nodes: Variation Nodes

- Mutate
- Mutate w/o creating new solution (Tweak)
- Uniform Recombination
- Diagonal Recombination
Nodes: Selection Nodes

- k-Tournament
- Truncation

Nodes: Set Nodes

- makeSet
- Persistent
Nodes: Set Nodes

- makeSet
- Persistent Sets
- addSet
- ‘Last’ Set

Nodes: Evaluation Node

- Evaluates the nodes passed in
- Allows multiple operations and accurate selections within an iteration
Meta-Genetic Program

- Create Valid Population
- Select Survivors
- Check Termination
- Generate Children
- Evaluate Children

BBSA Evaluation

- Translation
- Mutation rate = 0.5
- Add Set
- Make Set
- Tournament
- Last
- A
Termination Conditions

• Evaluations
• Iterations
• Operations
• Convergence

Testing

• Deceptive Trap Problem

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

![Graph showing fitness vs. number of bits]

Fitness vs. # of bits
Testing (cont.)

• Problem Configuration
  – Bit-length = 100
  – Trap Size = 5

• Verification Problem Configurations
  – Bit-length = 100, Trap Size = 5
  – Bit-length = 200, Trap Size = 5
  – Bit-length = 105, Trap Size = 7
  – Bit-length = 210, Trap Size = 7

• External Verification

Results

60% Success Rate
Results:
Bit-Length = 100
Trap Size = 5

Results:
Bit-Length = 200
Trap Size = 5
Results:
Bit-Length = 105
Trap Size = 7

Results:
Bit-Length = 210
Trap Size = 7
Over-Specialization

Trained Problem Configuration

Alternate Problem Configuration

BBSA2

Add Set

Evaluation

Diagonal Recombination

$n = 3$

Mutation rate = 99.3%

Truncation Count = 17

‘Last’

Truncation Count = 15

‘Last’

Truncation Count = 8

‘Last’
Summary

- Created novel meta-GP approach for evolving BBSAs tuned to specific problem classes
- Ideal for solving repeated problems
- Evolved custom BBSA which outperformed standard EA and hill-climber on all tested problem instances
- Future work includes adding additional primitives and testing against state-of-the-art BBSAs on more challenging problems

Take Home Message

- Practitioners need automated algorithm selection & configuration
- The meta-heuristical black-box optimization tapestry is getting steadily richer, but less navigatable
- Recent additions to the tapestry facilitate automated real-world problem solving
References


