Overview

- Rationale
- Swarm Intelligence Overview
- Swarm Simulation Software
- Evolutionary Computing for Swarms
- Results
- Conclusions
Motivations

- Interactions and complexity
- Complexity can grow quickly
- Swarm algorithm development requires simulation
- Trial-and-error programming
- A complete toolchain is needed to streamline swarm algorithm creation
  - Simulator
  - Algorithm generator
  - Identify/classify emergence
Vision

Demonstrate a method for generating swarm behaviors using evolutionary computing.
Swarm Intelligence

Examples of *Swarm Intelligence* found in nature

- Flocking birds
  - A bird flying disrupts airflow
  - Disrupted air flow reduces drag for following birds
  - Reduced drag results in easier flying
  - Distance traveled by the flock is maximized
Swarm Intelligence

Examples of *Swarm Intelligence* found in nature

- Foraging ants
  - An ant leaves a pheromone trail upon finding food
  - Other ants follow and reinforce the trail
  - Each ant is able to find food for the nest
  - Trail laying finds the closest food source
Swarm Intelligence

Examples of *Swarm Intelligence* found in nature

- Termite nests
  - A termite deposits a pheromone-tagged mud ball
  - Local pheromones affect mud ball placement
  - A secure nest for the termite is established
  - A temperature regulated nest emerges
Swarm Intelligence

Why has evolution produced swarming in so many different contexts?

- Simultaneously benefits the individual and the whole
- Individuals benefit from the efforts of others
- The survivability of the swarm increases
- Simple rules and behaviors, decentralized
- Replication relatively easy
Swarm Intelligence

Swarm intelligence as defined for this work

a group of agents whose collective interactions magnify the effects of individual agent behaviors, resulting in the manifestation of swarm level behaviors beyond the capability of a small subgroup of agents

Other properties required for emergent behavior

- Large numbers of agent interactions
- Ability to modify the environment, stigmergy
- Randomness
Swarm Algorithm Development

- No methods for direct analysis of swarm algorithms
- Swarm algorithms evaluated through simulation

Thus, a flexible swarm simulation platform is required.
- Multiple agent and swarm types
- Support for various environment types
- Access to all simulation data
- Easy to use
- Portable
S W E E P

S W E E P- S W a r m E x p e r i m e n t a t i o n a n d E v a l u a t i o n P l a t f o r m

Entity

Agent
Controller Actions Avatar Model Sensors

Environment

Environment

Environment

Probes

Swarm Algorithms – p. 9
**Sweep - Simulation**

- XML simulation specification file
- Responsible for constructing the simulation
- Handles update scheduling
  1. Environment
  2. `Entity::Agent`
  3. `Entity::Avatar`
  4. Probes

```xml
<simulation>
  <main/>
  <agent/>
  <controller/>
  <model/>
  <environment/>
  <probes/>
</simulation>
```
The “mind” of the `Entity`

- **State**: collection of variables that define the agent
- **Controller**: defines the governing logic of the agent
- **The default Controller** is a finite state machine

![Finite State Machine Diagram]

- `s-1`: 
  - Condition = 101
  - Action = esc

- `s-2`: 
  - Condition = 111
  - Action = jmp
  - Condition = 000
  - Action = add
The “body” of the Entity

- Conduit between Agents and Environments
- Separates modeling and algorithm development
- e.g., a UAV

- **Model**: defines characteristics
  e.g., minimum turning radius, maximum thrust
- **Sensor**: defines environmental information available
  e.g., chemical sensor, GPS
- **Action**: defines behavioral abilities
  e.g., plan-path-to, return-to-base
The Environment has three core functionalities:

1. Defining fundamental laws that Avatars must respect
   e.g., gravity, $F = ma$, bandwidth limits

2. Presenting an information abstraction layer
   e.g., neighborhood on a grid vs. a graph

3. Facilitating direct and indirect communication
   e.g., simulating wireless, pheromone gradients
Probes provide the ability to

- Extract information from a running simulation
- Inject information into a running simulation

The current Probe implementation uses Connectors.

- Connectors are data conduits between components
- Probes “tap” Connectors
- Connectors provide access to information injection/extraction

Example Probe usage: diagnostic interface
SWEEP Applications

This thesis:
- Dispersion
- Task assignment, CAST Auction
- Chemical cloud tracking

Other works:
- Swarm reasoning for the four-color mapping problem
- Mars exploration using “tumbleweeds”
- Extending swarm programming with aspect-oriented programming
UAV Chemical Cloud Tracking

Highlights

- Develop decentralized algorithms for small collections of UAVs
- Constrained vehicles
  - Limited communication
  - Limited flight capabilities
  - Binary sensors
- Explore potential emergent behavior of small collections of agents
UAV Chemical Cloud Tracking
UAV Chemical Cloud Tracking

Cloud begins to drift

Contaminated UAV

Dump Site
UAV Chemical Cloud Tracking
UAV Chemical Cloud Tracking

UAV Characteristics

- UAV flight model
  - Point mass
    \[
    \begin{align*}
    \dot{x} &= v_C \cos(\theta) \\
    \dot{y} &= v_C \sin(\theta) \\
    \dot{\theta} &= \omega
    \end{align*}
    \]
  - Fixed turning radius
  - Constant speed
  - Geometric path planner
- 30 minute power supply
- Binary chemical sensor
**UAV Chemical Cloud Tracking**

Data Collection

- Cloud detection only
- 20,000 simulations
  - 5 to 15 agents
  - 20 cloud sizes
- 30 minute power supply
- 40 knots constant speed
- Constant wind speed and direction

```plaintext
generateRandomPath()
loop
    if endOfPath() or rand() < .1
        then
            generateRandomFlightPlan()
        end if
end if
end loop
```
UAV Chemical Cloud Tracking

- Normalized search time
- Speed increases with swarm and cloud size
- Larger swarms are faster
- Bigger clouds are easier
UAV Chemical Cloud Tracking

- Detection rate
- Rate increases with swarm and cloud size
- Larger swarms are more accurate
- More UAVs, more area covered
Autogenerating Swarm Behaviors

Why autogenerate swarm behaviors?
- Trial-and-error gets tedious
- Complexity can quickly increase
- Emergence not always obvious
- Move away from low-level swarm programming
Autogenerating Swarm Behaviors

Why autogenerate swarm behaviors?
- Trial-and-error gets tedious
- Complexity can quickly increase
- Emergence not always obvious
- Move away from low-level swarm programming

What is needed?
- Specify high-level goals
- Define lower-level behaviors, sensors, ...
- Use simulation to evaluate performance
ECS Overview
# ECS - System Parameters

## Parameters

<table>
<thead>
<tr>
<th>Objective</th>
<th>Dispersion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. Generations</td>
<td>500</td>
</tr>
<tr>
<td>Population Size</td>
<td>32</td>
</tr>
<tr>
<td>Number of Sims</td>
<td>2</td>
</tr>
</tbody>
</table>

## Mutations

<table>
<thead>
<tr>
<th>Change-Sensor-Value</th>
<th>Add-Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>top 6 + 2 random</td>
<td>top 6 + 2 random</td>
</tr>
</tbody>
</table>

## Actions

<table>
<thead>
<tr>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>too-many neighbors</td>
</tr>
<tr>
<td>chemical-present</td>
</tr>
</tbody>
</table>

## Simulation

<table>
<thead>
<tr>
<th>Number of Agents</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>50 × 50 grid</td>
</tr>
</tbody>
</table>
**ECS - Solution Representation**

**SWEEP XML state machine**
- Simple but expressive
- Graph-based
- Robust to random modification

```xml
<states>
  <state name="A">
    <transition nextState="s-1">
      <sensor name="holding-object" value="true"/>
      <action name="jmp"/>
    </transition>
    ...
  </state>
  ...
</states>
```

- \(s_1\) condition = 101, action = esc
- \(s_2\) condition = 000, action = add
ECS - Mutations

- AddState
- AddTransition
- ChangeNextState
- InvertSensor
- ChangeAction
ECS - Mutations

- AddState
- AddTransition
- ChangeNextState
- InvertSensor
- ChangeAction
ECS - Mutations

- AddState
- AddTransition
- ChangeNextState
- InvertSensor
- ChangeAction
ECS - Mutations

- AddState
- AddTransition
- ChangeNextState
- InvertSensor
- ChangeAction
ECS - Mutations

- AddState
- AddTransition
- ChangeNextState
- InvertSensor
- ChangeAction
ECS - Fitness Evaluation

- Differentiate good and bad solutions, imposes order
- Solutions simulated in SWEEP
  - One state machine solution → single agent program
  - Homogeneous swarm
- Multiple runs, remove biases
- Calculate fitness from raw SWEEP output
- Error proportional to fitness, normalized

Example

\[
\begin{align*}
\text{sweep}(s_1) &= 7 & \text{sweep}(s_2) &= 12 \\
\text{error}(s_1) &= 20 - 7 = 13 & \text{error}(s_2) &= 20 - 12 = 8 \\
\text{fitness}(s_1) &= 13/20 = 0.65 & \text{fitness}(s_2) &= 12/20 = 0.40
\end{align*}
\]

s_2 more fit
Evolving Swarm Algorithms

Four scenarios examined:

- Agent Dispersion
- Object Collection
- Object Destruction
- Object Manipulation

Simultaneous object collection and destruction
Object Manipulation

Two types of objects

- \( C \rightarrow \) objects to be collected
- \( D \rightarrow \) objects to be destroyed

Swarm Goal

- Collect all \( C \) objects
- Destroy all \( D \) objects

Approach

1. Collection
2. Destruction
3. Collection and Destruction
## Object Manipulation

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Collection</td>
</tr>
<tr>
<td>move-up</td>
<td>x</td>
</tr>
<tr>
<td>move-down</td>
<td>x</td>
</tr>
<tr>
<td>move-left</td>
<td>x</td>
</tr>
<tr>
<td>move-right</td>
<td>x</td>
</tr>
<tr>
<td>move-random</td>
<td>x</td>
</tr>
<tr>
<td>pick-up</td>
<td>x</td>
</tr>
<tr>
<td>put-down</td>
<td>x</td>
</tr>
<tr>
<td>move-to-goal</td>
<td>x</td>
</tr>
<tr>
<td>broadcast_C</td>
<td>x</td>
</tr>
<tr>
<td>move-to-object_C</td>
<td>x</td>
</tr>
<tr>
<td>first-attack</td>
<td></td>
</tr>
<tr>
<td>second-attack</td>
<td></td>
</tr>
<tr>
<td>broadcast_D</td>
<td></td>
</tr>
<tr>
<td>move-to-object_D</td>
<td></td>
</tr>
</tbody>
</table>
## Object Manipulation

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Collection</th>
<th>Destruction</th>
<th>Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>near-object_C</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>on-object_C</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>holding-object_C</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>on-goal</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>near-object_D</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>on-object_D(untouched)</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>on-object_D(damaged)</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
## Object Manipulation

<table>
<thead>
<tr>
<th>Fitness Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>number of objects picked up but not put in the goal</td>
</tr>
<tr>
<td>$c_2$</td>
<td>number of objects not collected</td>
</tr>
<tr>
<td>$d_1$</td>
<td>number of objects in the partially destroyed state</td>
</tr>
<tr>
<td>$d_2$</td>
<td>number of objects in the untouched state</td>
</tr>
<tr>
<td>$t$</td>
<td>number of time steps</td>
</tr>
</tbody>
</table>
Object Manipulation

The challenge

Collection and destruction metrics are independent but equally weighted

- Imposing an order skews evolution
- “Experts” are evolved

Solution
  - Construct new composite metrics
  - Eliminate sequential dependencies
  - Rank solutions using radix-based sorting
## Object Manipulation

<table>
<thead>
<tr>
<th>Composite Metric</th>
<th>Composition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>$\sim c_1 \land \sim d_1$</td>
<td>flag, fully performing both</td>
</tr>
<tr>
<td>$m_2$</td>
<td>$\sim c_2 \land \sim d_2$</td>
<td>flag, partially performing both</td>
</tr>
<tr>
<td>$m_3$</td>
<td>$\sim c_1 \lor \sim d_1$</td>
<td>flag, fully performing either</td>
</tr>
<tr>
<td>$m_4$</td>
<td>$\sim c_2 \lor \sim d_2$</td>
<td>flag, partially performing either</td>
</tr>
<tr>
<td>$m_5$</td>
<td>$\max(c_1, d_1)$</td>
<td>select the weakest</td>
</tr>
<tr>
<td>$m_6$</td>
<td>$\max(c_2, d_2)$</td>
<td>select the weakest</td>
</tr>
<tr>
<td>$m_7$</td>
<td>$\min(c_1, d_1)$</td>
<td>select the strongest</td>
</tr>
<tr>
<td>$m_8$</td>
<td>$\min(c_2, d_2)$</td>
<td>select the strongest</td>
</tr>
<tr>
<td>$m_9$</td>
<td>$t$</td>
<td>number of timesteps</td>
</tr>
</tbody>
</table>
Object Manipulation

Parameters
- Population: 32
- Mutations: all top 6 + 2 random
- SWEEP: 100 agents, $50 \times 50$ grid
- $C$ objects = 50
- $D$ objects = 30
- Broadcast range = 25
- Sensing range = 5
Best fitness score for Object Manipulation

Swarm Algorithms – p. 36
Object Manipulation

Mean fitness score for Object Manipulation

Normalized Error

Generation
Object Manipulation

Best fitness score for Object Manipulation

Normalized Error vs Generation

Swarm Algorithms – p. 37
Object Manipulation
Object Manipulation

Distribution of metric $m_s$ scores over each generation

Solution

Generation

Swarm Algorithms – p. 38
Object Manipulation

Distribution of metric $m$ scores over each generation

Solution

Generation
Conclusions

- Re-designed and Implemented SWEEP
  - Demonstrated the capabilities of SWEEP
  - Better suited for larger/more complex problems
  - Successfully used in applications outside this work
- Designed and implemented ECS
- Established the feasibility of evolving state machines for swarm algorithms
- Successfully generated swarm algorithms for a number of different scenarios
- Demonstrated the use of composite metrics and radix-based ranking to address multi-objective problems
Future Work

- More efficient SWEEP core
- Build a standard SWEEP component library
- Use aspect-oriented programming for probing
- Explore other solution encodings
- Attempt more difficult problems
- Autogenerate composite metrics from high-level goals
- Methods of detecting / measuring emergent behaviors
Acknowledgments

Advisor(s):
- Dr. Michael Branicky
- Dr. Dan Palmer
- Dr. Ravi Vaidyanathan

Committee:
- Dr. Randall Beer
- Dr. Roger Quinn

Also:
- Orbital Research, Inc.
Questions
Swarm Intelligence

Definitions

- Beni, Hackwood, and Wang

  Unintelligent agents with limited processing capabilities, but possessing behaviors that collectively are intelligent
Swarm Intelligence

Definitions

Bonabeau, Dorigo, and Theraluz

Any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies.
Swarm Intelligence

Definitions

- Clough

A collection of simple autonomous agents that depend on local sensing and reactive behaviors to emerge global behaviors.
Swarm Intelligence

Benefits of swarm intelligence
- Robust
- Distributed
- Parallel
- Simple agents
- Scalability
- Effort Magnification
UAV Chemical Cloud Tracking

Searching
UAV Chemical Cloud Tracking

Mapping
Object Manipulation

Best fitness score for Object Manipulation

Normalized Error vs Generation

Swarm Algorithms – p. 48
Object Manipulation

Mean fitness score for Object Manipulation

Swarm Algorithms – p. 48
Dispersion

- Swarm Goal: achieve a density level
- Agent Goal:
  - neighbor at least $d_{\text{min}}$ units away
  - neighbor not more than $d_{\text{max}}$ units away
Dispersion

- Fitness: Number of agents violating dispersion criteria

<table>
<thead>
<tr>
<th>Actions</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>move-up</td>
<td>neighbor-above</td>
</tr>
<tr>
<td>move-down</td>
<td>neighbor-below</td>
</tr>
<tr>
<td>move-left</td>
<td>neighbor-left</td>
</tr>
<tr>
<td>move-right</td>
<td>neighbor-right</td>
</tr>
</tbody>
</table>

- Population: 32
- Mutations: all top 6 + 2 random
- **Sweep**: 100 agents, $50 \times 50$ grid
- $d_{min} = 2$, $d_{max} = 4$, range = 6
Dispersion

Best and average fitness score for Dispersion

- **Best**
- **Average**