Evolving swarm intelligence for task allocation in a real time strategy game

SBGAMES 2014

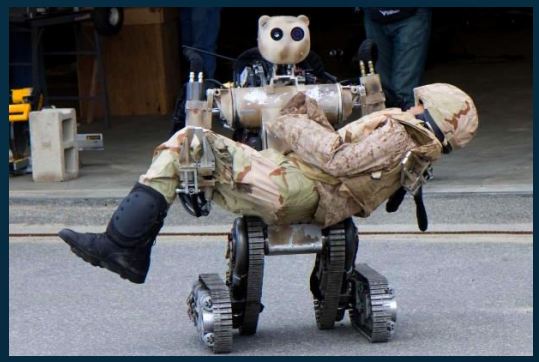
VERLab
Visão e Robótica

UFMG

ANDERSON TAVARES
HÉCTOR AZPÚRUA
LUIZ CHAIMOWICZ

The problem

- ***Coordination*** in complex scenarios
  - Multiple agents
  - Partial observability
  - Dynamic environment

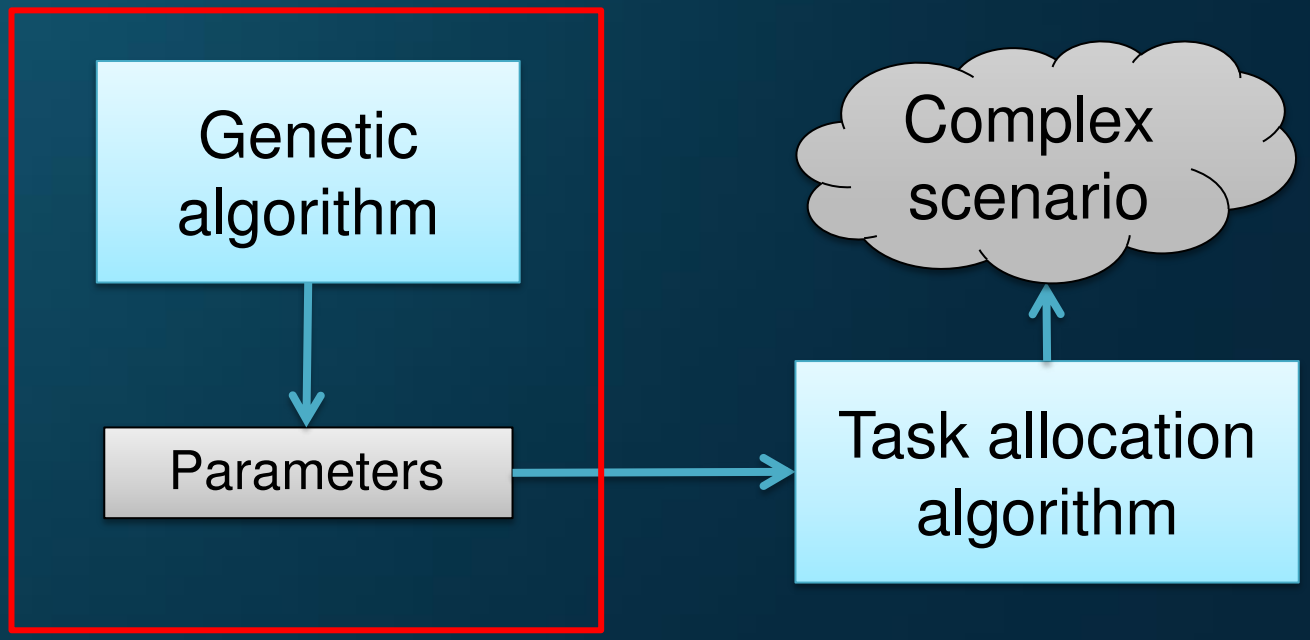


Rescue in disasters

- ***Coordination*** → task allocation
  - Divide goal into tasks
  - Assign tasks to agents

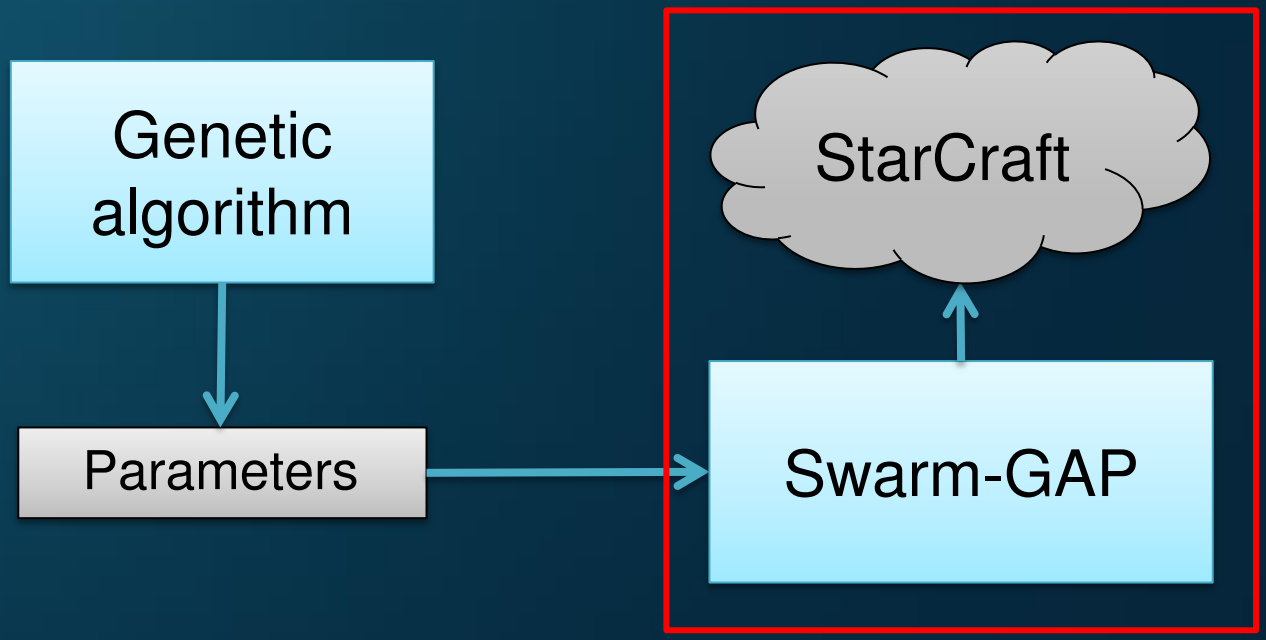
RTS game (StarCraft)

Our approach - Goals



- Automatically adjust task allocation parameters

Our approach - Goals



- Automatically adjust task allocation parameters
- Employ task allocation in an RTS game (StarCraft)

Related work – Task allocation

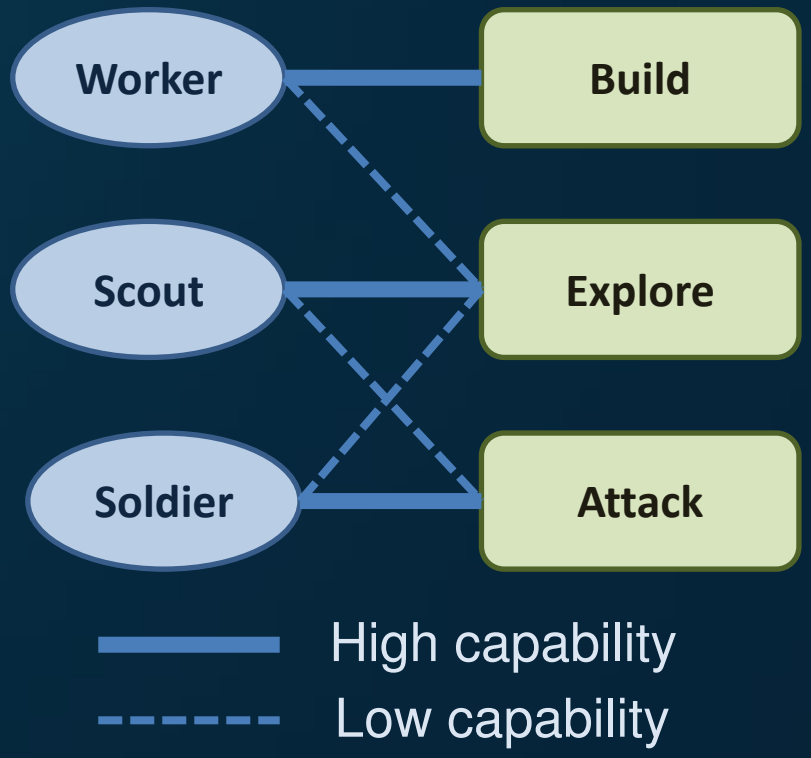
- Many algorithms for task allocation
  - LA-DCOP [1]
  - Branch-and-Bound Fast-Max-Sum [2]
  - *many others!*

- *But parameters are configured by hand*

[1] Scerri et al, “Allocating tasks in extreme teams,” in AAMAS, 2005.
[2] Macarthur et al, “A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems,” in AAAI, 2011.

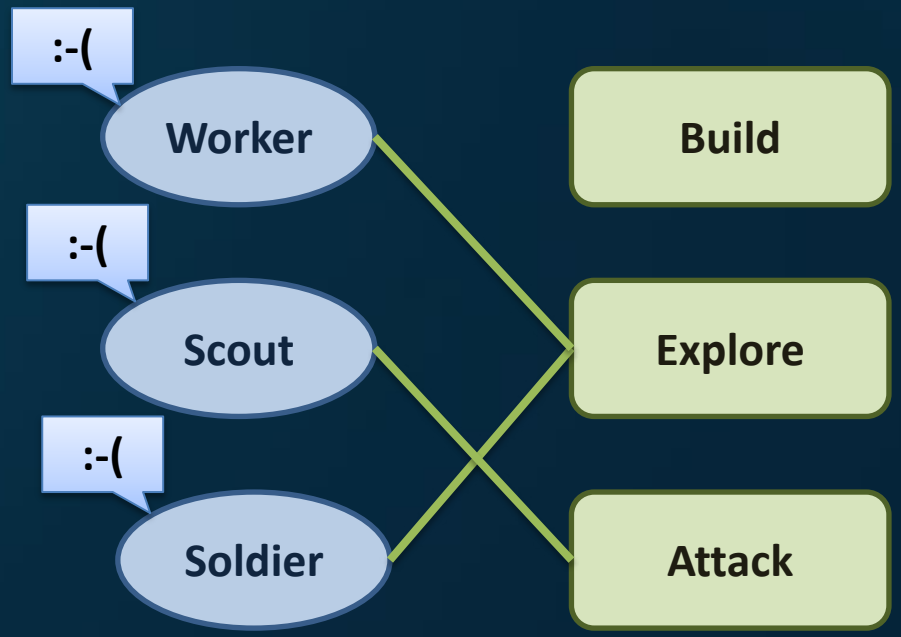
Task allocation

- *An optimization problem...*
- Given:
  - A set of tasks
  - A set of agents
    - (and their capabilities)



Task allocation

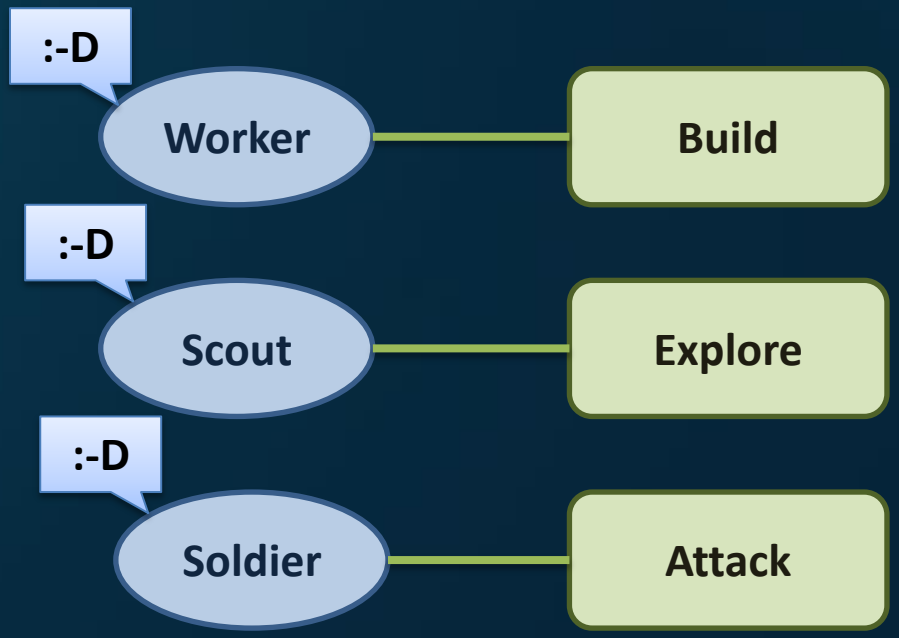
- *An optimization problem...*
- Given:
  - A set of tasks
  - A set of agents
    - (and their capabilities)
- Find:
  - The best task-agent assignment
  - *Utility* given by agent-task compatibility
- NP-Complete!



(a *bad* assignment)

Task allocation

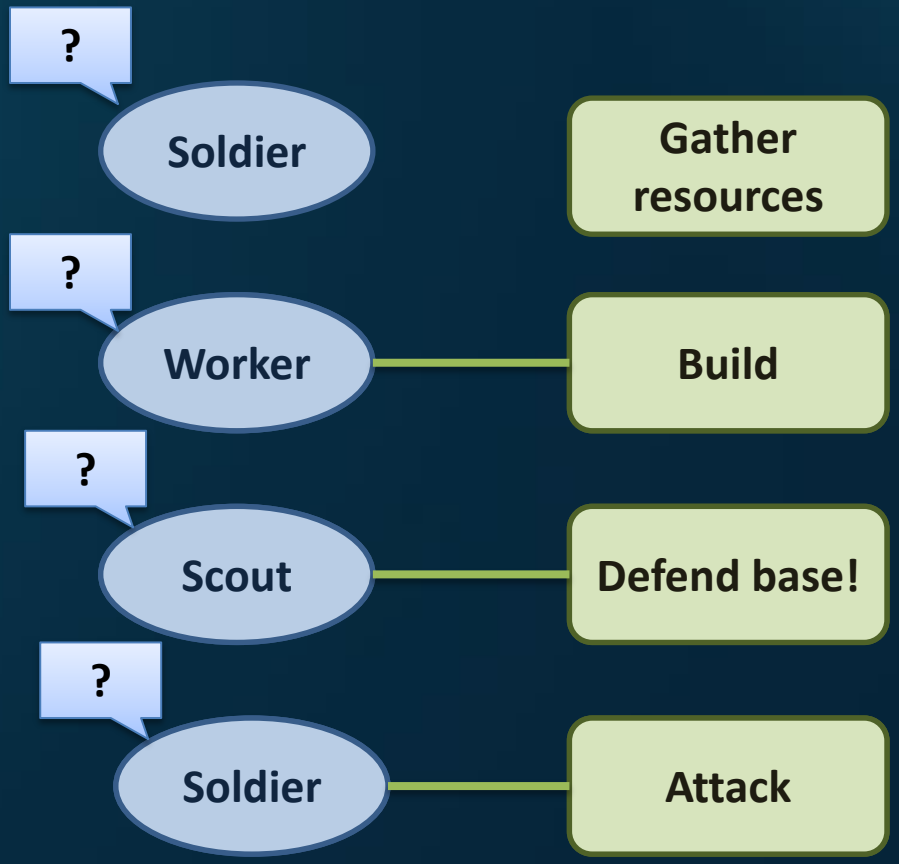
- *An optimization problem...*
- Given:
  - A set of tasks
  - A set of agents
    - (and their capabilities)
- Find:
  - The best task-agent assignment
  - *Utility* given by agent-task compatibility
- NP-Complete!



(a *good* assignment)

Task allocation

- Complex scenarios
  - Environment changes
  - Must reassign tasks
  - We need scalability and robustness

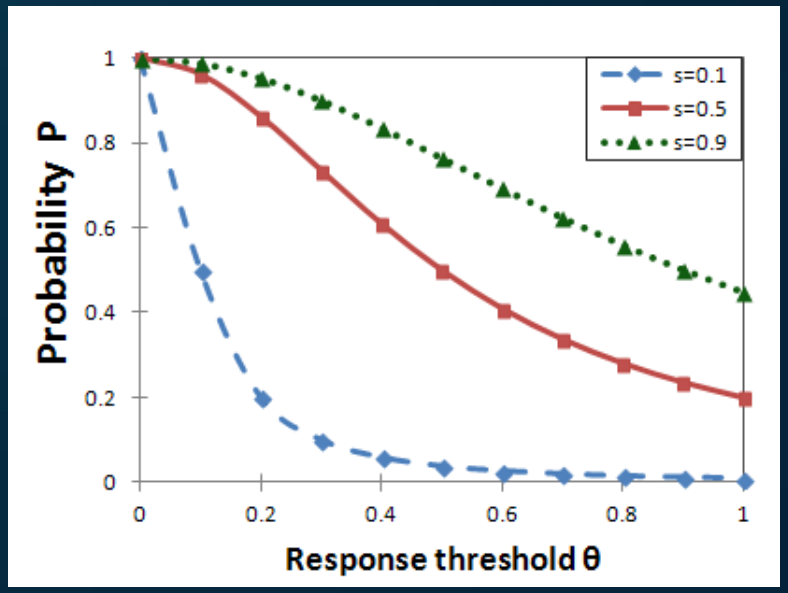


(now what?)

Swarm-GAP[1]

- Tasks have associated stimuli (s)
- Agents have response thresholds to tasks (θ)
- Probability to engage in task depends on both:

$$P(s,\theta) = \frac{s^2}{s^2 + \theta^2}$$



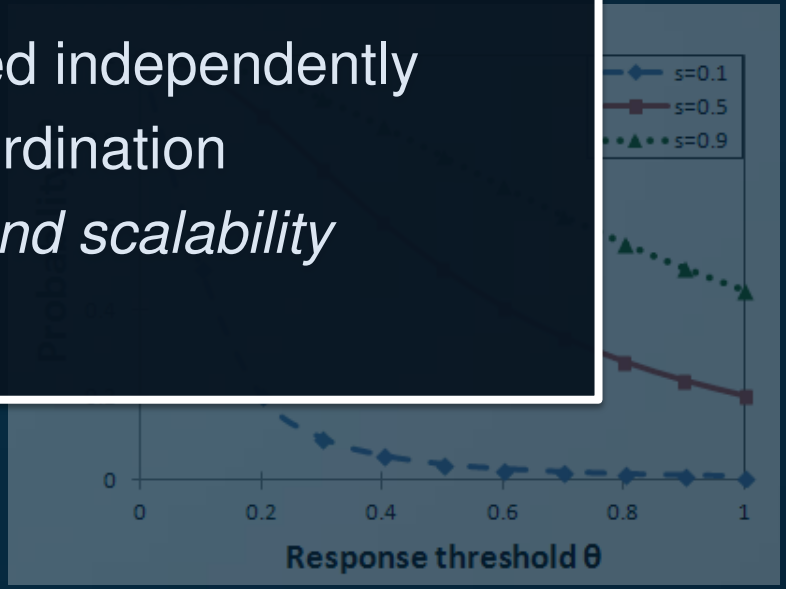
[1] Ferreira et. Al. Using Swarm-GAP for distributed task allocation in complex scenarios. Massively Multiagent Systems. 2008

Swarm-GAP[1]

- Tasks have associated stimuli (s)
- Agents have response thresholds to tasks (θ)
- Probability to engage in task depends on both:

Strengths of Swarm-GAP

- Tasks allocated independently
- Emergent coordination
- *Robustness and scalability*



[1] Ferreira et. Al. Using Swarm-GAP for distributed task allocation in complex scenarios. Massively Multiagent Systems. 2008

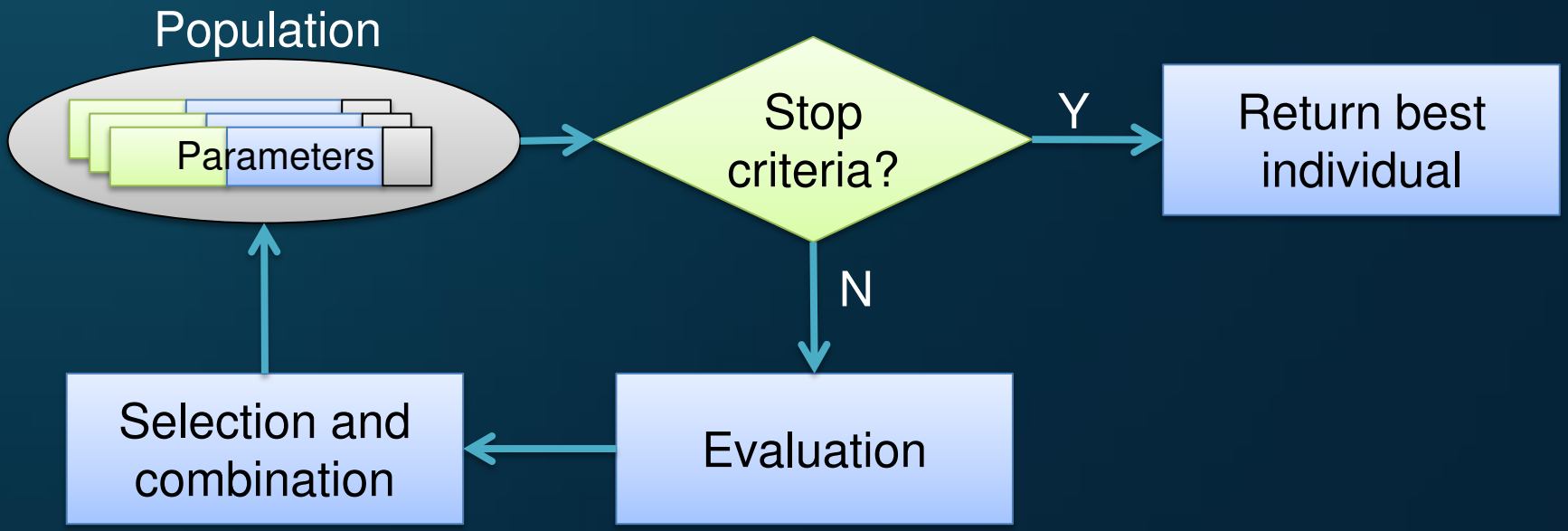
StarCraft – our testbed

- Popular RTS game with 3 races:
  - Terran
  - Zerg
  - Protoss
- In-game score based on:
  - Resource management
  - Base expansion
  - Attack and defense
- Our bot implements Swarm-GAP
  - Plays with Terran
  - Uses 7 out of 17 buildings
  - Uses 3 out of 13 units

Terran Zerg Protoss

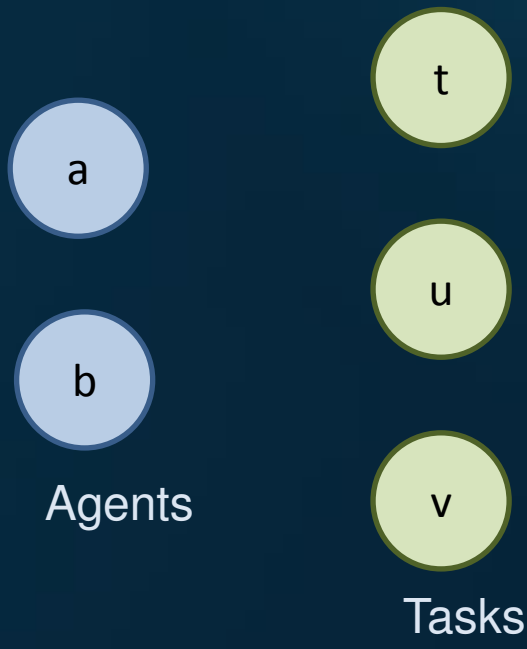
Evolving Swarm-GAP

- The genetic algorithm
  - An individual is an array of Swarm-GAP parameters



Evolving Swarm-GAP

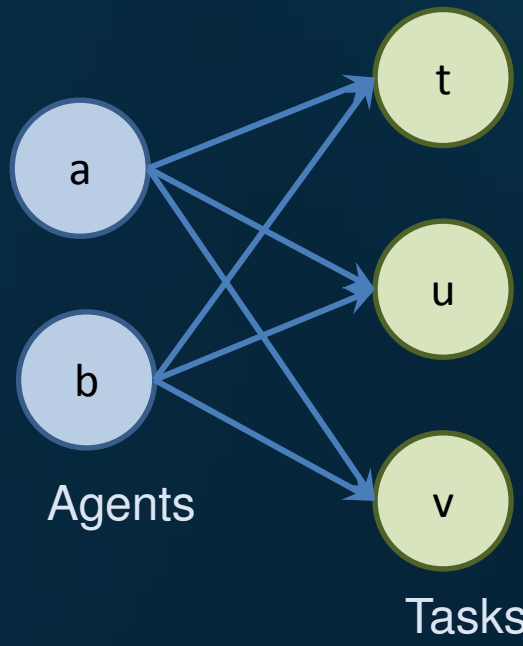
- Array of parameters:
  - Stimuli for each task



s_t	s_u	s_v

Evolving Swarm-GAP

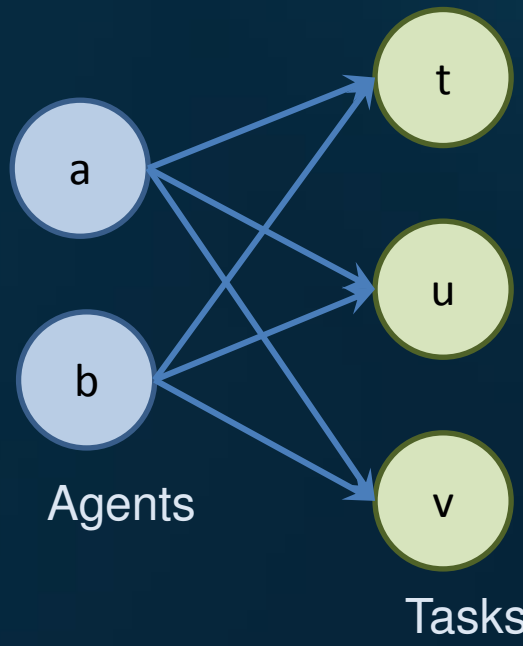
- Array of parameters:
  - Stimuli for each task
  - Capability for each agent-task combination



s_t	s_u	s_v	k_{at}	k_{au}	k_{av}	k_{bt}	k_{bu}	k_{bv}

Evolving Swarm-GAP

- Array of parameters:
  - Stimuli for each task
  - Capability for each agent-task combination
  - One game-related parameter
    - Army size



s_t	s_u	s_v	k_{at}	k_{au}	k_{av}	k_{bt}	k_{bu}	k_{bv}	g

Evolving Swarm-GAP

- $Fitness = \frac{our\ bot's\ score}{opponent's\ score}$

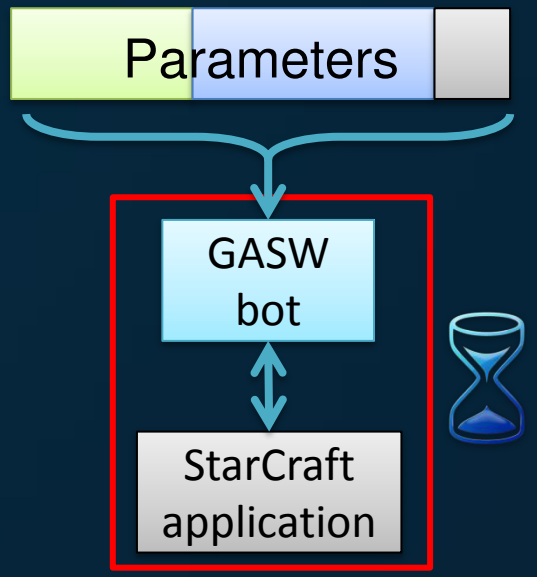
If fitness > 1: our bot won the match

Evolving Swarm-GAP

- $Fitness = \frac{our\ bot's\ score}{opponent's\ score}$

If fitness > 1: our bot won the match

- Problem:
  - Evaluation depends on match results
  - Time-consuming!
- Solution
  - *Estimate* fitness of some individuals
  - "Interpolate" parents' fitness

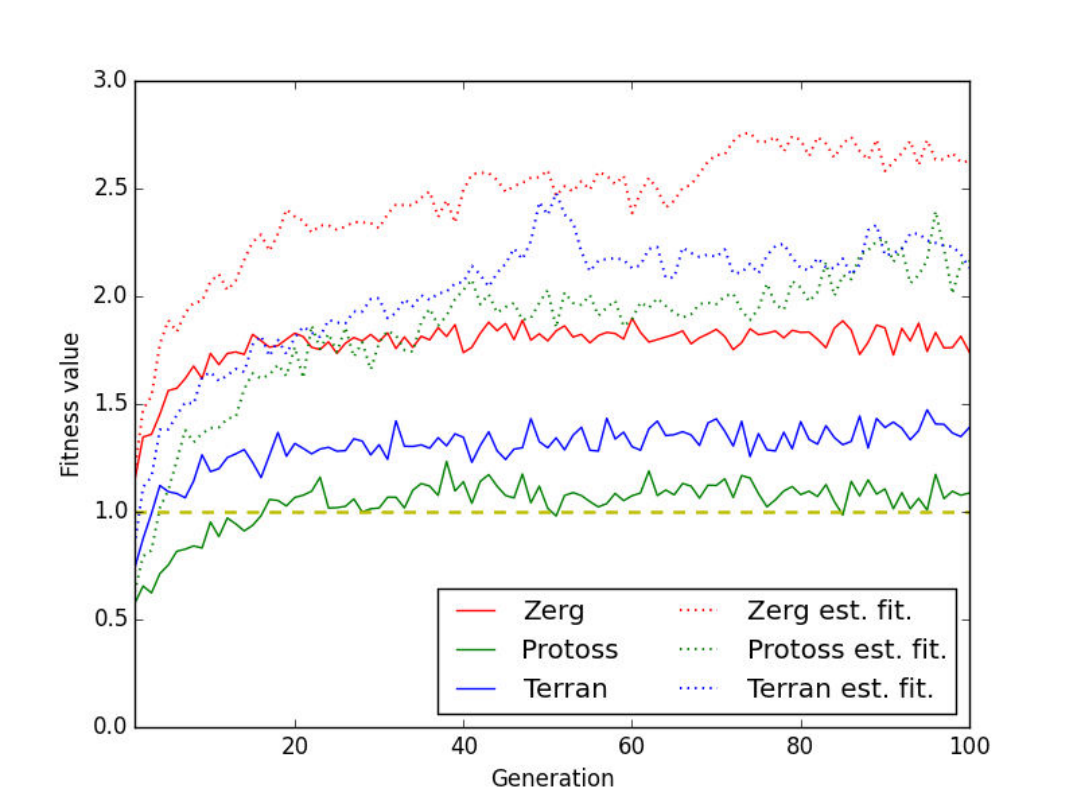


Experiments

1. Evaluate GA behavior
   - Fitness along generations
   - Evaluation vs. estimation

2. Compare different approaches
   - Victory rate against StarCraft's native AI
   - Validation of our approach

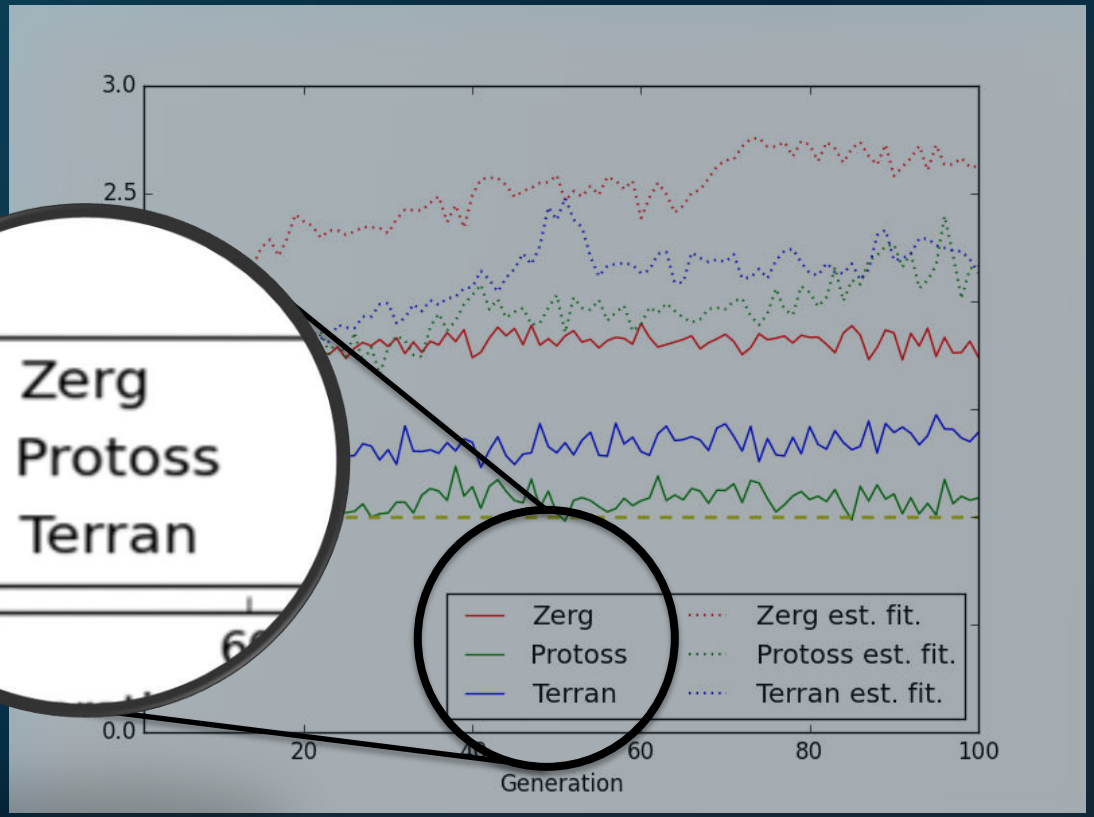
Experiment 1 – GA behavior



Mean fitness per generation

Experiment 1 – GA behavior

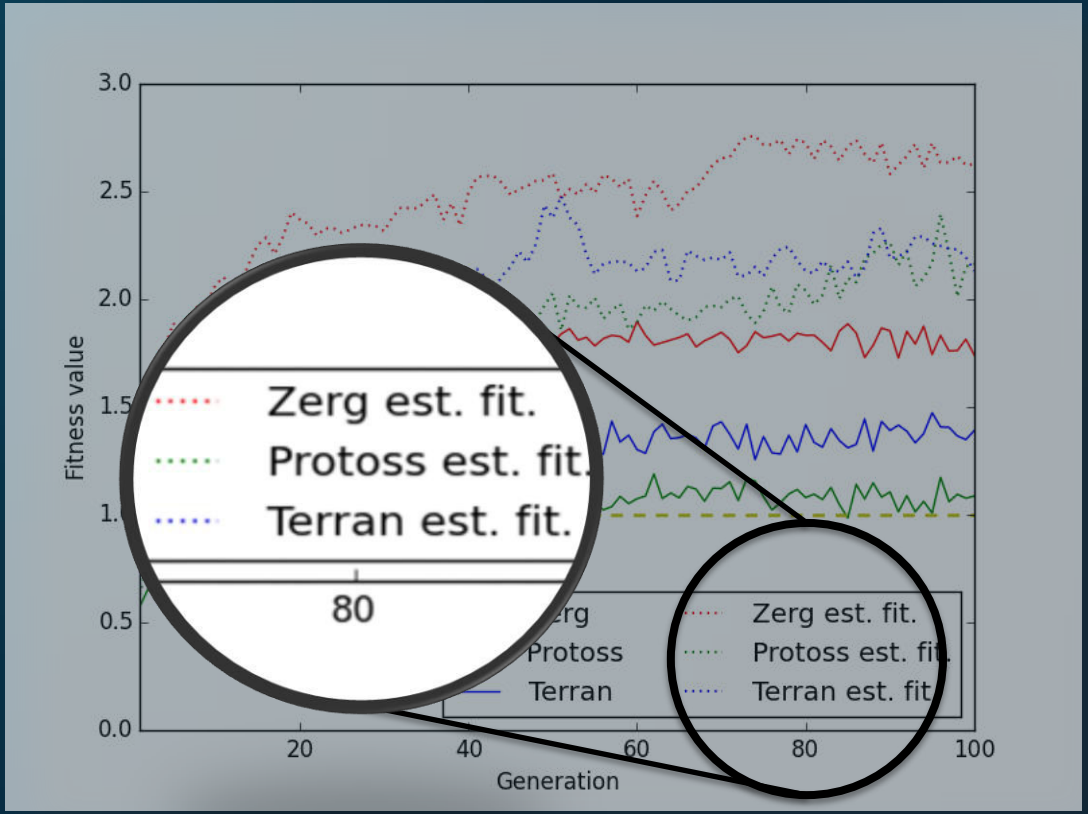
Results with fitness evaluation.



Mean fitness per generation

Experiment 1 – GA behavior

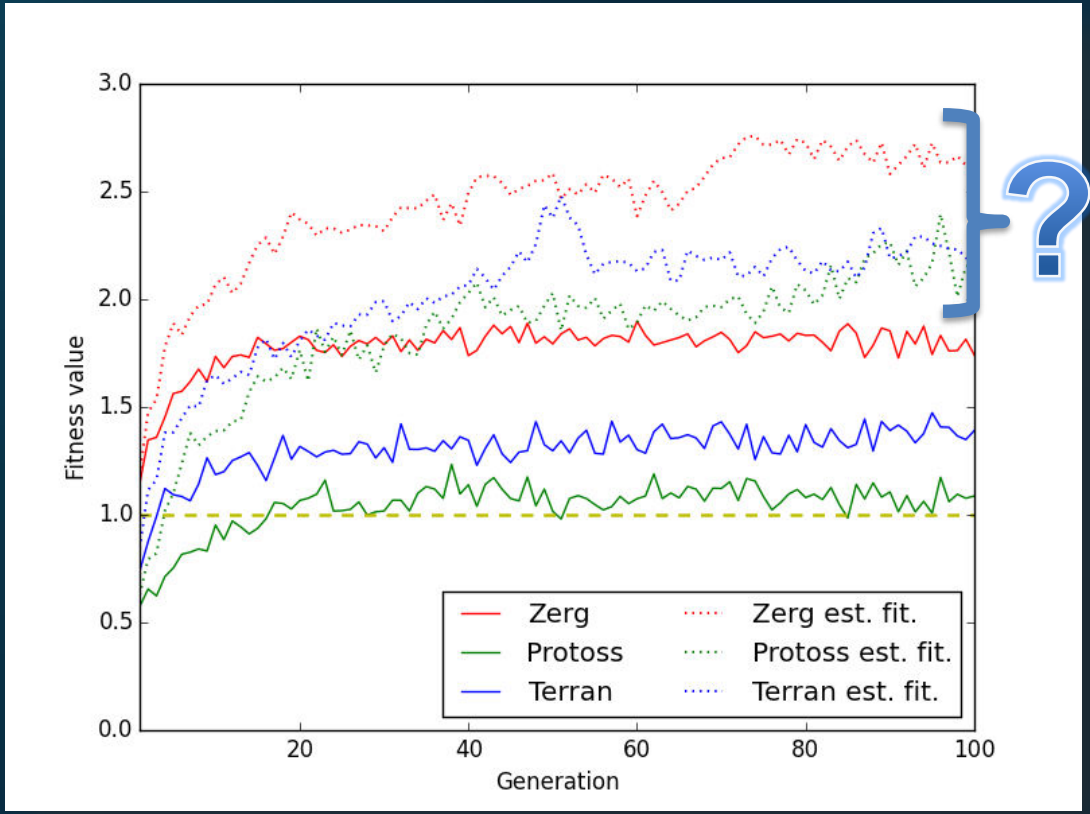
Results with fitness estimation.



Mean fitness per generation

Experiment 1 – GA behavior

- Superior fitness with estimation
- Is this reliable? Wait for part 2!



Mean fitness per generation

Experiment 2 - with other bots

1. GASW:
   - Best parameters found without fitness estimation
2. GASW-e:
   - Best parameters found with fitness estimation
3. ManSW:
   - Hand-configured array of parameters.
4. Random bot:
   - For each agent, a task is chosen with uniform probability.
5. AIUR:
   - Competition bot, placed 3rd in AIIDE 2013[1] and CIG 2013[2].

[1] http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/report2013.shtml
[2] http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2013

Experiment 2 - with other bots

1. GASW:
   - Best parameters found without fitness estimation
2. GASW-e:
   - [illegible]
3. ManSW:
   - [illegible]
4. Random bot:
   - For each agent a task is chosen with uniform probability
5. AIUR:
   - Competition bot, placed 3rd in AIIDE 2013[1] and CIG 2013[2].

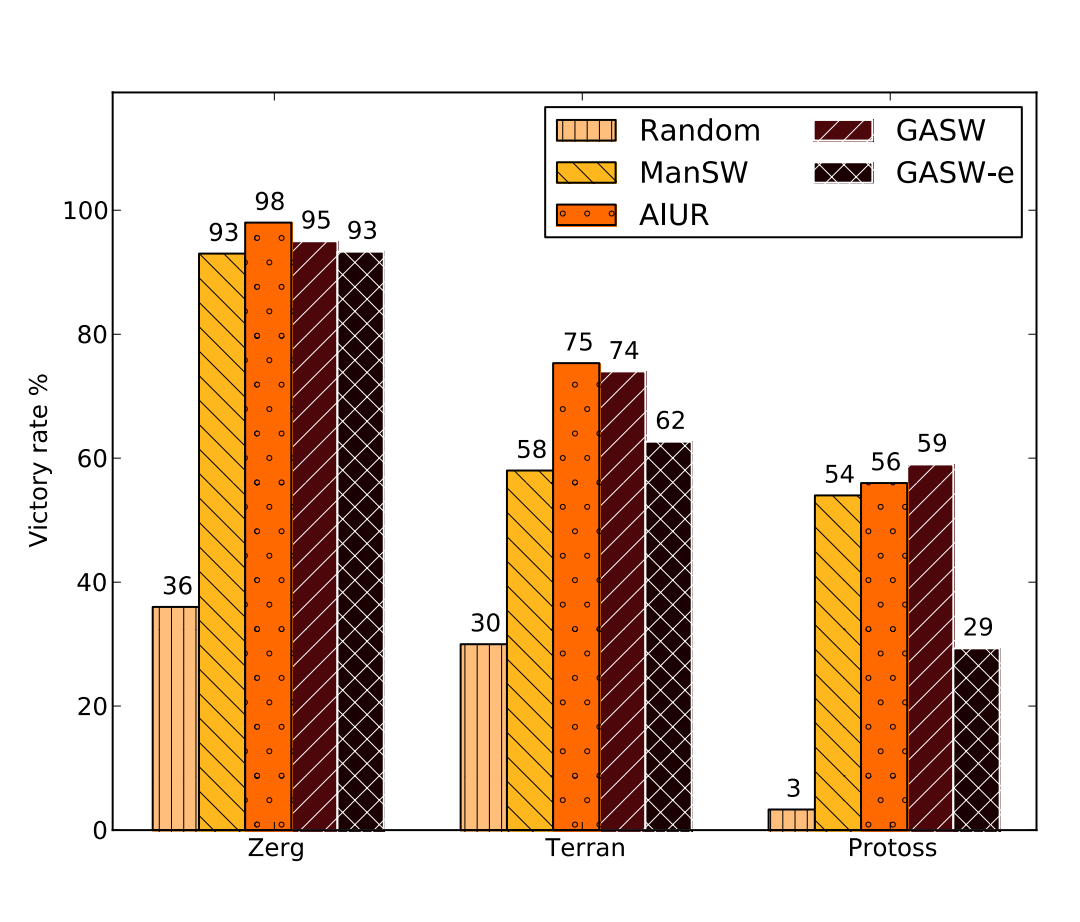
ManSW , GASW and GASW-e:

- Tasks allocated via Swarm-GAP
- Difference: parameter configuration
  - This will validate our approach

[1] http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/report2013.shtml
[2] http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2013

VERLab Visão e Robótica

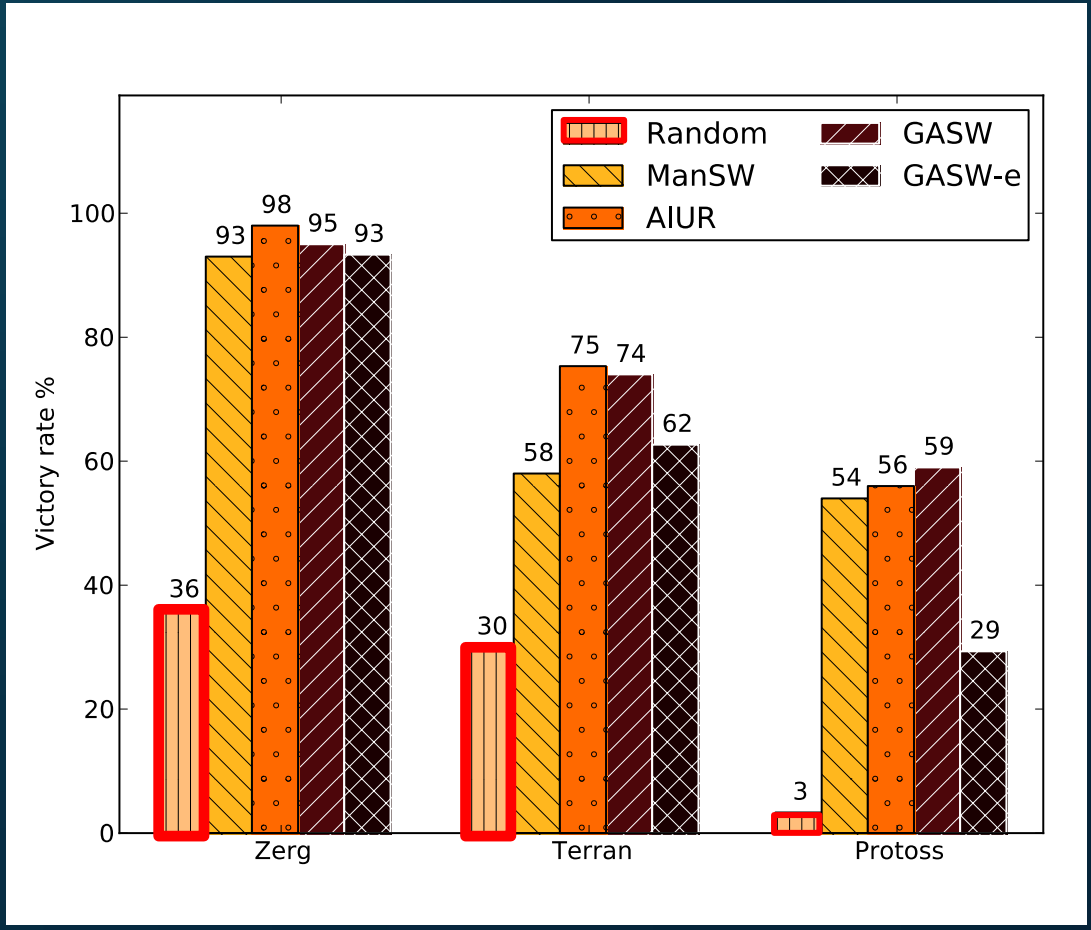
Experiment 2 - with other bots



Victory rate on 150 matches against StarCraft's Native AI

Experiment 2 - with other bots

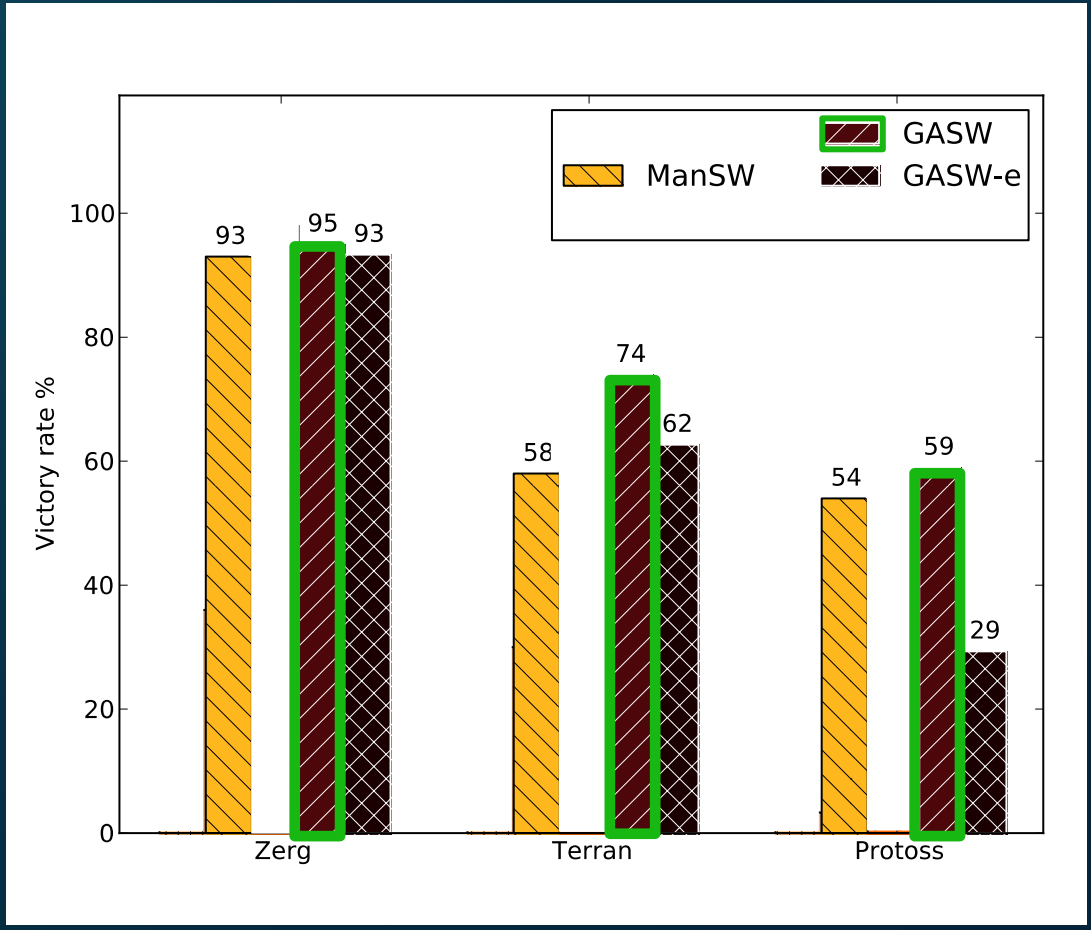
- Random has the worst performance



Victory rate on 150 matches against StarCraft's Native AI

Experiment 2 - with other bots

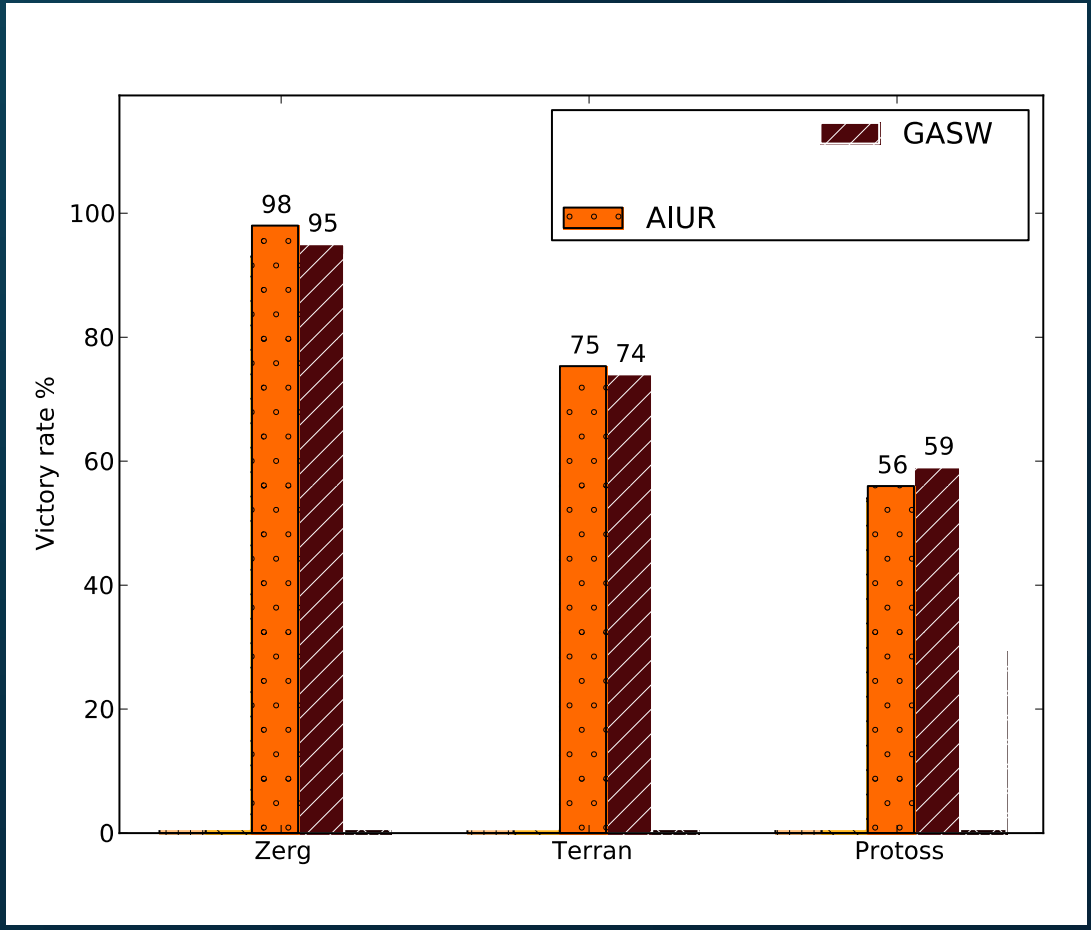
- Random has the worst performance
- GASW outperforms GASW-e and ManSW.



Victory rate on 150 matches against StarCraft's Native AI

Experiment 2 - with other bots

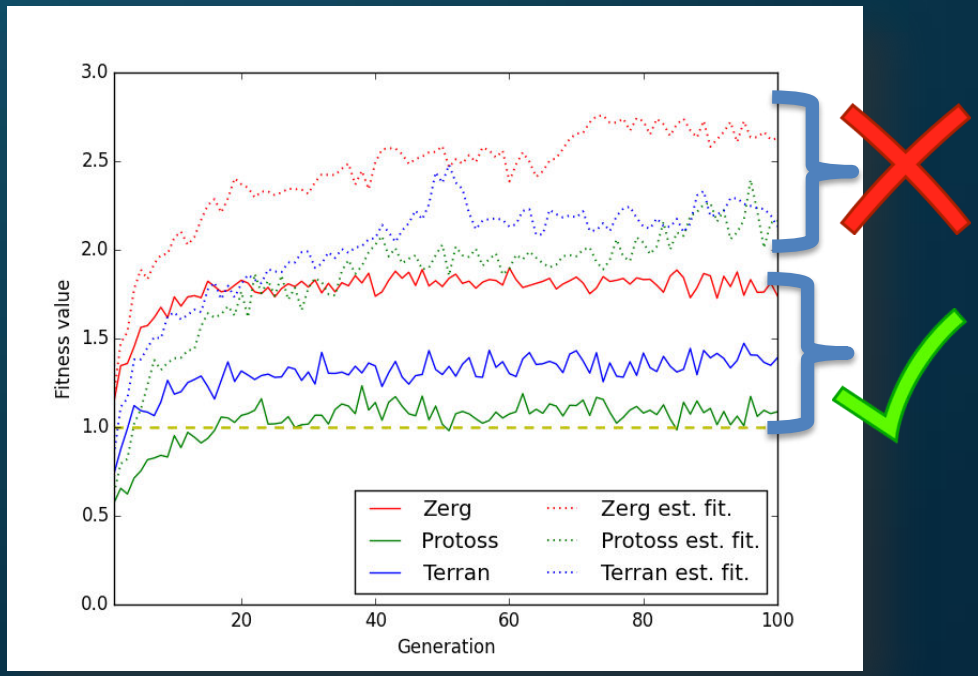
- Random has the worst performance
- GASW outperforms GASW-e and ManSW.
- GASW and AIUR achieve similar performance.



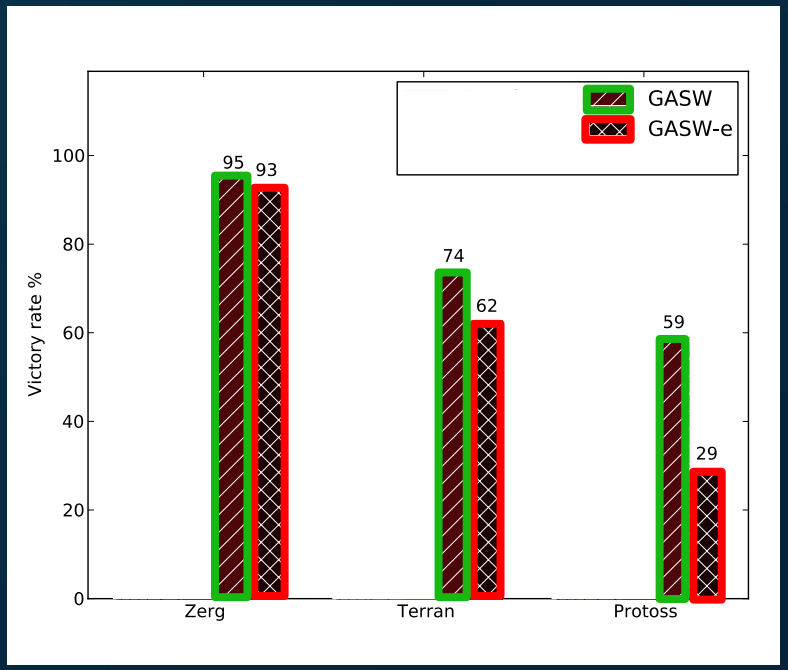
Victory rate on 150 matches against StarCraft's Native AI

Bottomline

- In our scenario, fitness estimation wasn't reliable



Fitness value per generation



Victory rate on 150 matches against StarCraft's Native AI

Bottomline

- Fitness estimation was misleading:
  - Fitness is noisy: opponent behavior and probabilistic task allocation.

- “Lucky” individual propagates itself with estimation
  - It brings the search to its neighborhood.
  - Which may not be the optimum region.

Conclusion

- Contributions:
  - Systematic approach to adjust parameters for task allocation in complex scenarios.
  - Evaluation of fitness estimation in a noisy environment.
- Promising results:
  - Random and manual were outperformed.
  - Victory rate at par with AIUR.
- However:
  - We couldn’t play direct matches vs AIUR :-(

Future work

- Improve fitness estimation:
  - Deal with fitness noise
  - Which conditions lead to reliable fitness estimation?

- Improve game performance:
  - Use all units and buildings
  - Opening book, micromanagement, terrain analysis...

The end

Evolving swarm intelligence for task allocation in a real time strategy game

anderson
hector.azpurua
chaimo
} @dcc.ufmg.br

Questions?

VERLab
Visão e Robótica

Appendix – E-GAP model

- "The math behind task allocation"
  - I: agents; J = tasks; a_{ij} = allocation indicator

$$W = \sum_{t} \sum_{i^t \in \mathcal{I}^t} \sum_{j^t \in \mathcal{J}^t} k_{ij} \times a_{ij}^t - \sum_{t} \sum_{j^t \in \mathcal{J}^t} (1 - a_{ij}^t) \times d_j^t \quad \text{(1a)}$$

$$\text{subject to:} \quad \forall t \forall i^t \in \mathcal{I}^t, \sum_{j^t \in \mathcal{J}^t} c_{ij}^t \times a_{ij}^t \leq r_i^t \quad \text{(1b)}$$

$$\text{and:} \quad \forall t \forall j^t \in \mathcal{J}^t, \sum_{i^t \in \mathcal{I}^t} a_{ij}^t \leq 1 \quad \text{(1c)}$$

Appendix - Swarm-GAP algorithm

- Initiate token (set of tasks)
- For each task j in token:
  - `If random()` $< P(s_j, \theta_j)$:
    - engage in task j
- Forward token with remaining tasks

All agents execute this algorithm

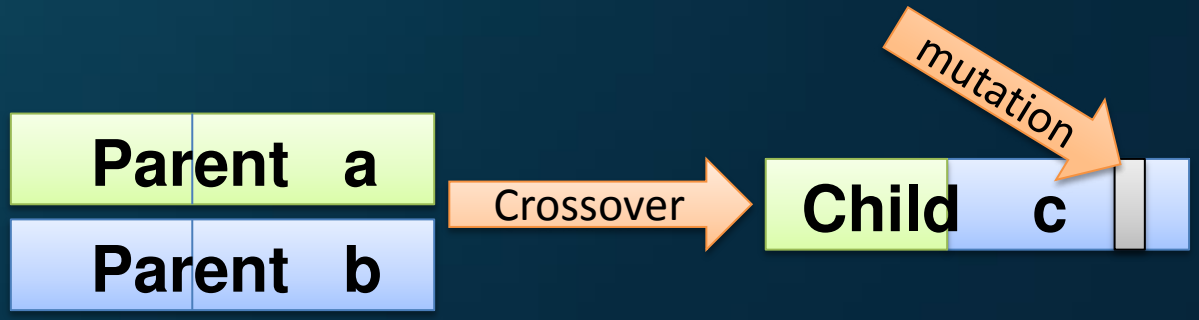
Appendix - Swarm-GAP in StarCraft

- Agent-task compatibility:

Task	SCV	Marine	Commander
Gather minerals	√		
Build barracks	√		
Build supply depot	√		
Build academy	√		
Build refinery	√		
Build command center	√		
Repair building	√		
Explore map	√	√	
Attack	√	√	
Train SCV			√
Train medic			√
Train marine			√

Appendix - Evolving Swarm-GAP

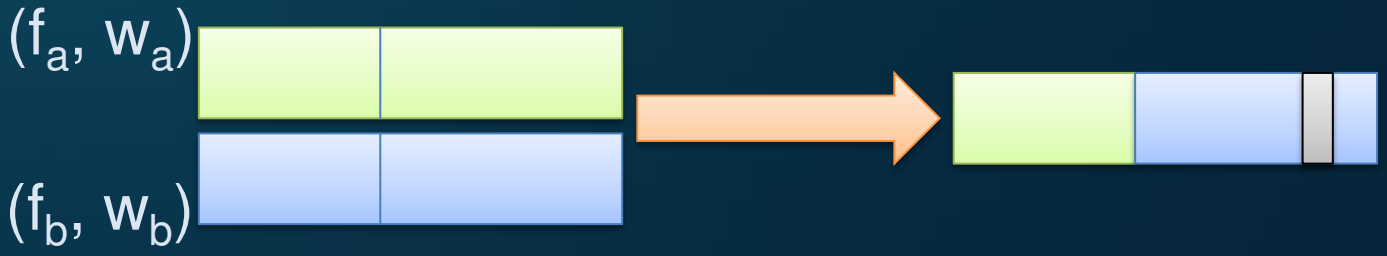
- Fitness estimation*
  - Parents generate offspring as usual



* **Method by [Salami and Hendtlass 2003]**

Appendix - Evolving Swarm-GAP

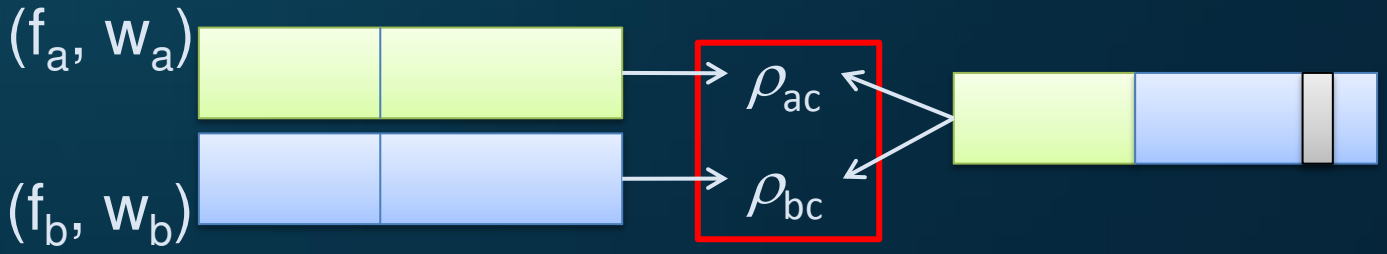
- Fitness estimation
  - Individuals have fitness value (f) and reliability (w)
  - When parents generate offspring:
    - Calculate parent-child similarity (ρ)
    - Estimate child fitness
    - Calculate child reliability



Reliability: 'how close' estimated f is to actual fitness

Appendix - Evolving Swarm-GAP

- Fitness estimation
  - Individuals have fitness value (f) and reliability (w)
  - When parents generate offspring:
    - Calculate parent-child similarity (ρ)
    - Estimate child fitness
    - Calculate child reliability

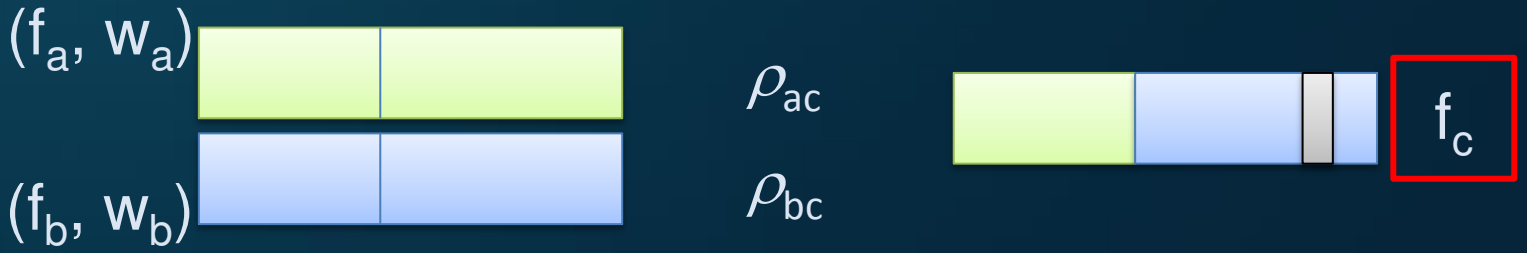


$$\rho_{ac} = 1 - \frac{1}{|A|}\sum_{i=1}^{|A|}\frac{abs(A[i] - C[i])}{max_i - min_i}$$

ρ_{xy} is a measure of "*distance*" of values in x and y

Appendix - Evolving Swarm-GAP

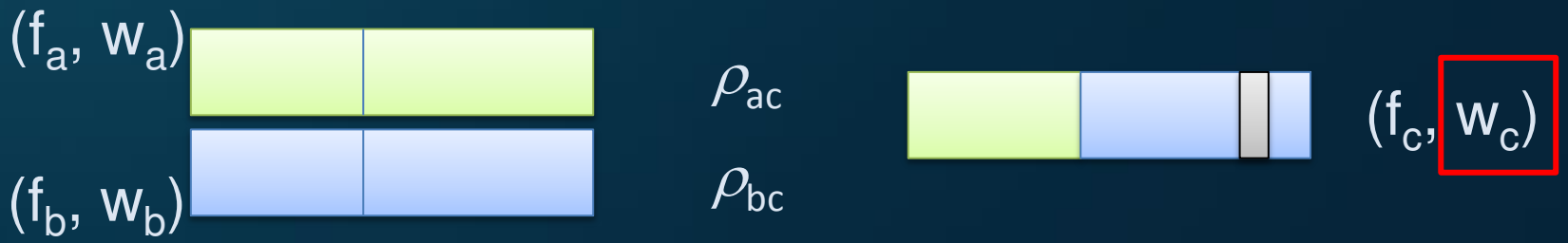
- Fitness estimation
  - Individuals have fitness value (f) and reliability (w)
  - When parents generate offspring:
    - Calculate parent-child similarity (ρ)
    - Estimate child fitness
    - Calculate child reliability



$$f_c = \frac{f_a w_a \rho_{ac} + f_b w_b \rho_{bc}}{w_a \rho_{ac} + w_b \rho_{bc}}$$

Appendix - Evolving Swarm-GAP

- Fitness estimation
  - Individuals have fitness value (f) and reliability (w)
  - When parents generate offspring:
    - Calculate parent-child similarity (ρ)
    - Estimate child fitness
    - Calculate child reliability



$$w_c = \frac{(w_a \rho_{ac})^2 + (w_b \rho_{bc})^2}{w_a \rho_{ac} + w_b \rho_{bc}}$$

Appendix - Evolving Swarm-GAP

- Fitness estimation
  - Individuals have fitness value (f) and reliability (w)
  - When parents generate offspring:
    - Calculate parent-child similarity (ρ)
    - Estimate child fitness
    - Calculate child reliability
    - Maintain reliability

(f_c, w_c)

If w_c < threshold: $f_c \leftarrow$ evaluate(c);
A few individuals are always evaluated.

Appendix - GA parameters

- Tournament selection (2 participants)
  - With elitism
- One-point crossover
- Crossover probability: 0.9
- Mutation probability: 0.01 per locus
- 100 generations
- 30 individuals