A Multi-Population Genetic Algorithm for UAV Path Re-Planning under Critical Situation

Jesimar S. Arantes (USP)
Márcio S. Arantes (USP)
Claudio F. M. Toledo (USP)
Brian C. Williams (MIT)

São Carlos, SP

November – 2015

Outline

Introduction

Overview

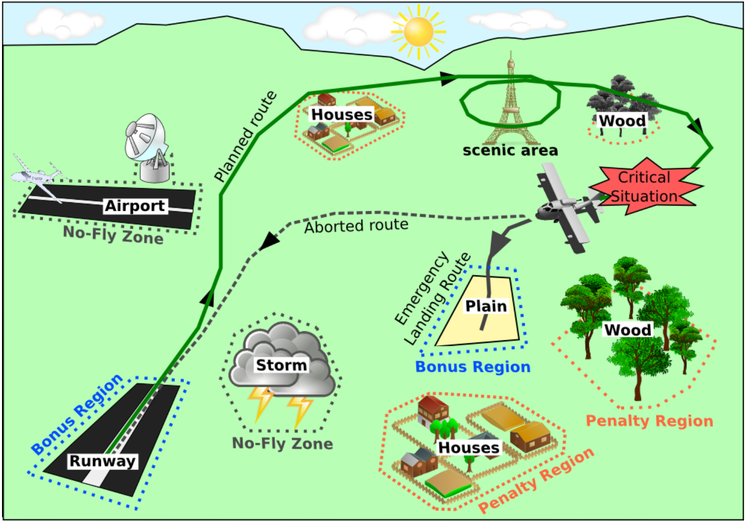


Figure 1: Illustrative scenario for mission planning.

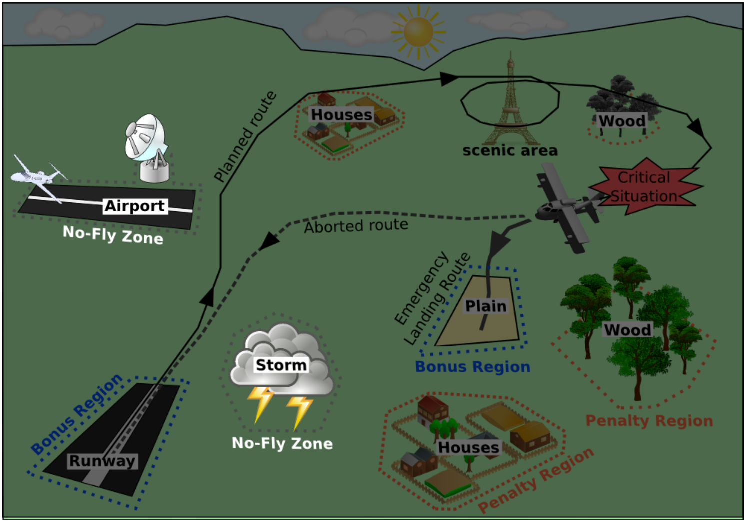


Figure 2: Illustrative scenario for mission planning.

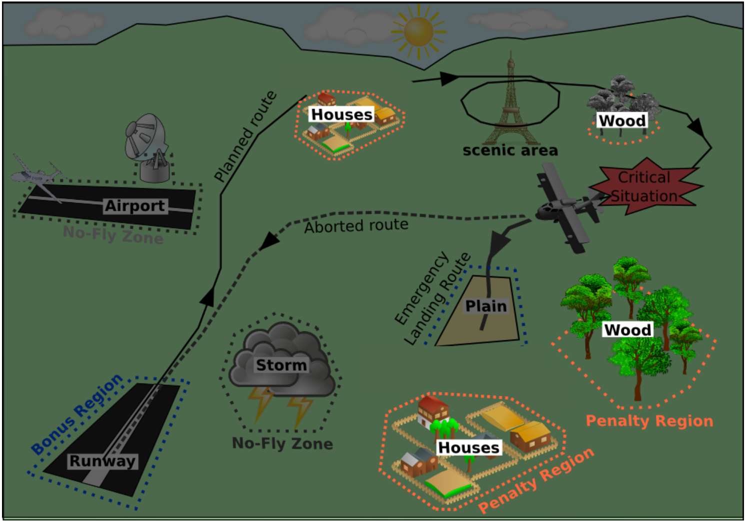


Figure 3: Illustrative scenario for mission planning.

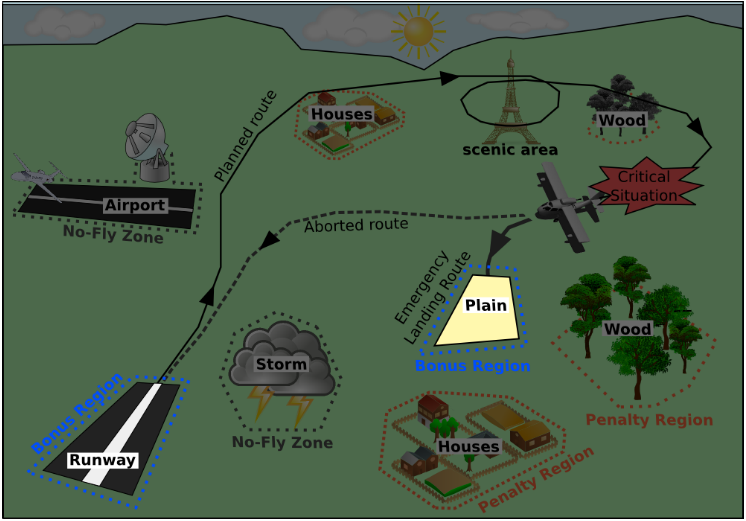


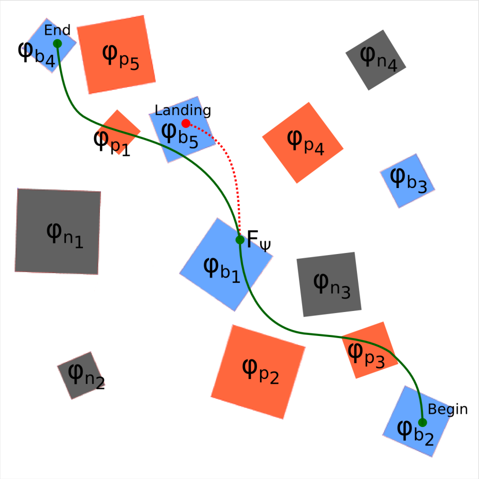
Figure 4: Illustrative scenario for mission planning.

Introduction
Overview

Figure 5: Illustrative scenario for mission planning.

Figure 6: Illustrative scenario for mission planning.

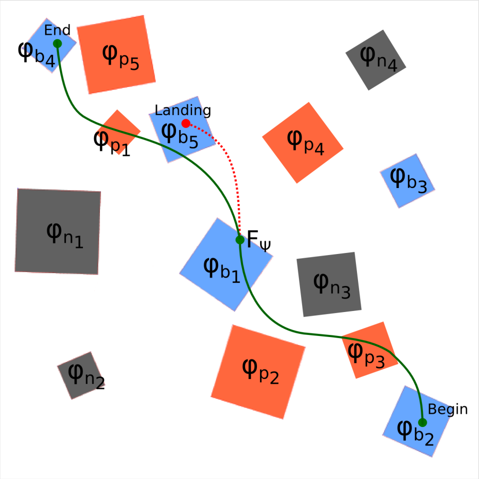
- Regions
  1. No-Fly Zone (ϕ_n) ■
  2. Penalty Region (ϕ_p) ■
  3. Bonus Region (ϕ_b) ■
  4. Remainder Region (ϕ_r) □
- Critical Situation
  1. Motor Failure (ψ_m)
  2. Battery Failure (ψ_b)
  3. Aerodynamic Surfaces Failure type 1 (ψ_{s1})
  4. Aerodynamic Surfaces Failure type 2 (ψ_{s2})



Problem Description

Types of Regions and Critical Situation

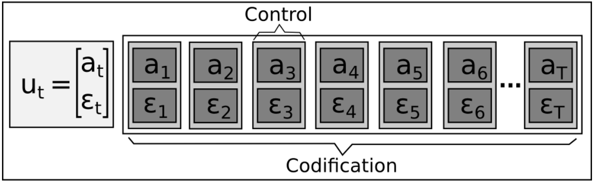
- Regions
  1. No-Fly Zone (ϕ_n) ■
  2. Penalty Region (ϕ_p) ■
  3. Bonus Region (ϕ_b) ■
  4. Remainder Region (ϕ_r) □
- Critical Situation
  1. Motor Failure (ψ_m)
  2. Battery Failure (ψ_b)
  3. Aerodynamic Surfaces Failure type 1 (ψ_{s^1})
  4. Aerodynamic Surfaces Failure type 2 (ψ_{s^2})



Methods

Codification, Decodification and Solution

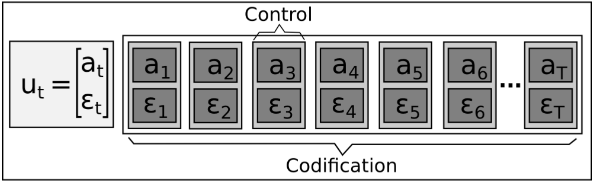
- Codification u_t:



Methods

Codification, Decodification and Solution

- Codification u_t:

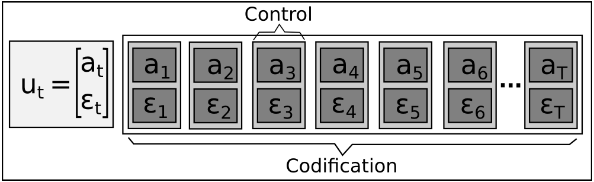


- Decodification F_Ψ:

$$x_{t+1} = F_\Psi(x_t, u_t)$$

$$\begin{bmatrix} p^x_{t+1} \\ p^y_{t+1} \\ v_{t+1} \\ \alpha_{t+1} \end{bmatrix} = \begin{bmatrix} p^x_t + v_t \cdot cos(\alpha_t) \cdot \Delta T + a_t \cdot cos(\alpha_t) \cdot (\Delta T)^2/2 \\ p^y_t + v_t \cdot sen(\alpha_t) \cdot \Delta T + a_t \cdot sen(\alpha_t) \cdot (\Delta T)^2/2 \\ v_t + a_t \cdot \Delta T - F^d_t \\ \alpha_t + \varepsilon_t \cdot \Delta T \end{bmatrix}$$

- Codification u_t:

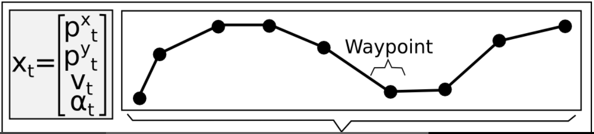


- Decodification F_Ψ:

$$x_{t+1} = F_\Psi(x_t, u_t)$$

$$\begin{bmatrix} p^x_{t+1} \\ p^y_{t+1} \\ v_{t+1} \\ \alpha_{t+1} \end{bmatrix} = \begin{bmatrix} p^x_t + v_t \cdot cos(\alpha_t) \cdot \Delta T + a_t \cdot cos(\alpha_t) \cdot (\Delta T)^2/2 \\ p^y_t + v_t \cdot sen(\alpha_t) \cdot \Delta T + a_t \cdot sen(\alpha_t) \cdot (\Delta T)^2/2 \\ v_t + a_t \cdot \Delta T - F^d_t \\ \alpha_t + \varepsilon_t \cdot \Delta T \end{bmatrix}$$

- Solution x_t:

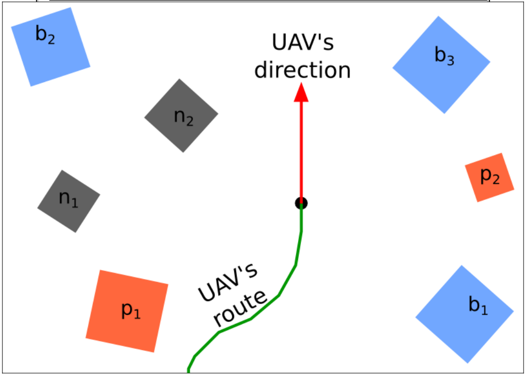


Methods

Greedy Heuristic

Algorithm 1: Greedy Heuristic.

begin
  RouteLanding *route*[] $\leftarrow$ RouteLanding()[*map*.$|\phi_b|$];
  for i = *1* **to** *map*.$|\phi_b|$ **do**
    initialize(*route*[i], *map*.$Z^i_{\phi_b}$);
    evaluate(*route*[i]);
  RouteLanding *bestRoute* $\leftarrow$ getBestRoute(*route*);
  return *bestRoute*;

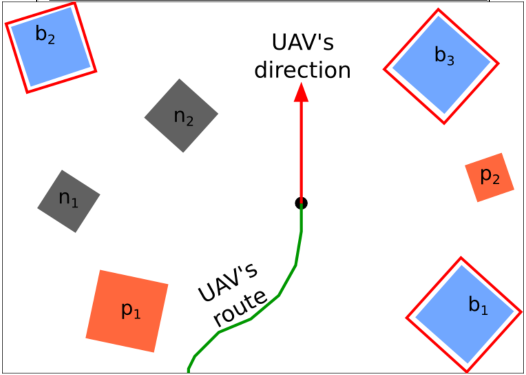


Methods

Greedy Heuristic

Algorithm 1: Greedy Heuristic.

begin
  RouteLanding *route*[] ← RouteLanding()[$map.|\phi_b|$];
  for $i = 1$ **to** $map.|\phi_b|$ **do**
    initialize(*route*[i], $map.Z^{i}_{\phi_b}$);
    evaluate(*route*[i]);
  RouteLanding *bestRoute* ← getBestRoute(*route*);
  return *bestRoute*;



Methods

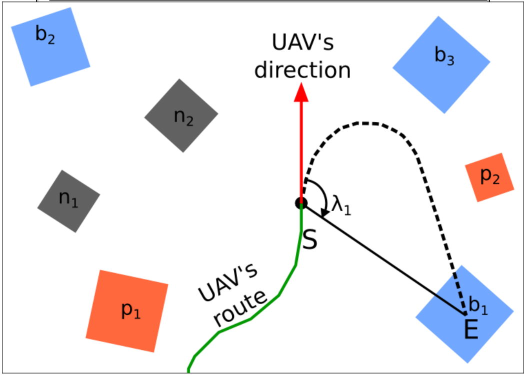
Greedy Heuristic

Algorithm 1: Greedy Heuristic.

```
begin
    RouteLanding route[] ← RouteLanding()[map.|φ_b|];
    for i = 1 to map.|φ_b| do
        initialize(route[i], map.Z^i_{φ_b});
        evaluate(route[i]);
    RouteLanding bestRoute ← getBestRoute(route);
    return bestRoute;
```

i = 1

$Z^1_{\varphi b} = b_1$

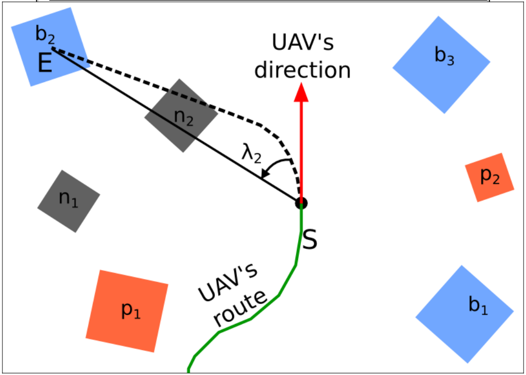


Algorithm 1: Greedy Heuristic.

begin
  RouteLanding *route*[] ← RouteLanding()[$map.|\phi_b|$];
  for i = *1* **to** $map.|\phi_b|$ **do**
    initialize(*route*[i], $map.Z^{i}_{\phi_b}$);
    evaluate(*route*[i]);
  RouteLanding *bestRoute* ← getBestRoute(*route*);
  return *bestRoute*;

i = 2

$Z^2_{\varphi b} = b_2$



Methods

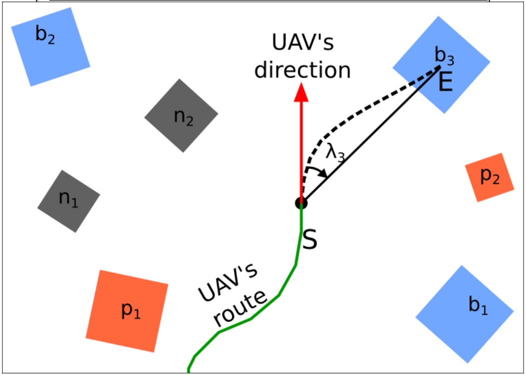
Greedy Heuristic

Algorithm 1: Greedy Heuristic.

```
begin
    RouteLanding route[] ← RouteLanding()[map.|φ_b|];
    for i = 1 to map.|φ_b| do
        initialize(route[i], map.Z^i_{φ_b});
        evaluate(route[i]);
    RouteLanding bestRoute ← getBestRoute(route);
    return bestRoute;
```

i = 3

$Z^3_{\varphi b} = b_3$



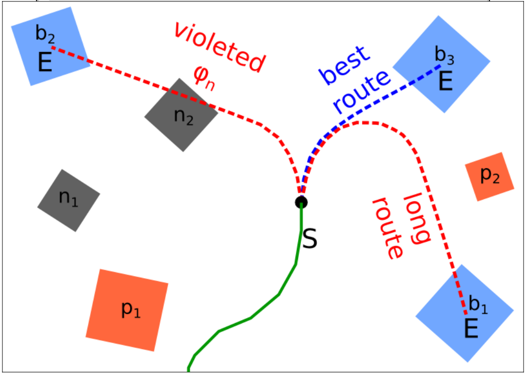
Methods

Greedy Heuristic

Algorithm 1: Greedy Heuristic.

```
begin
    RouteLanding route[] ← RouteLanding()[map.|ϕ_b|];
    for i = 1 to map.|ϕ_b| do
        initialize(route[i], map.Z^i_{ϕ_b});
        evaluate(route[i]);
    RouteLanding bestRoute ← getBestRoute(route);
    return bestRoute;
```

bestRoute = b_3

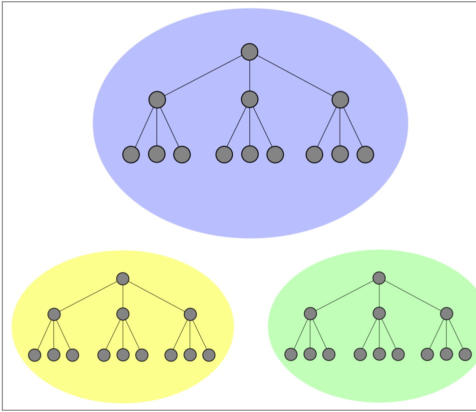


Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```



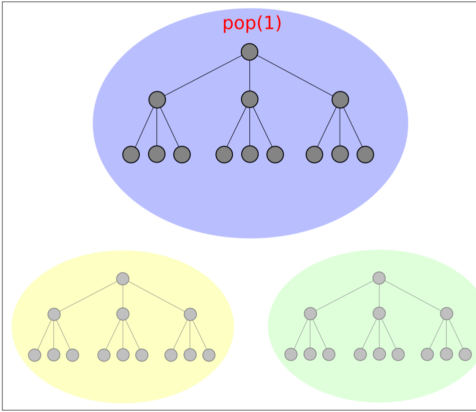
Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

i = 1



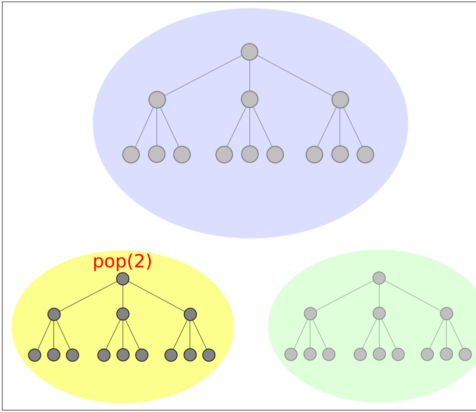
Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

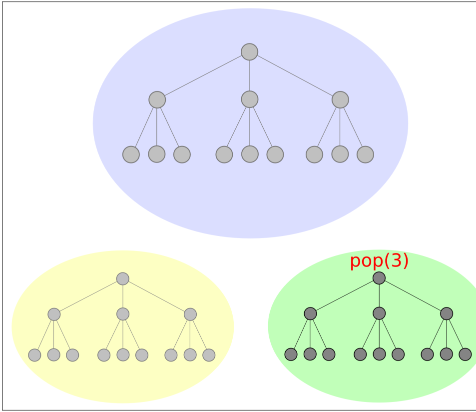
i = 2



Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

i = 3

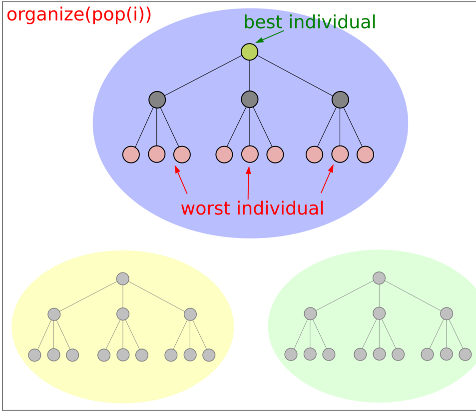


Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));                i = 1, 2, 3
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

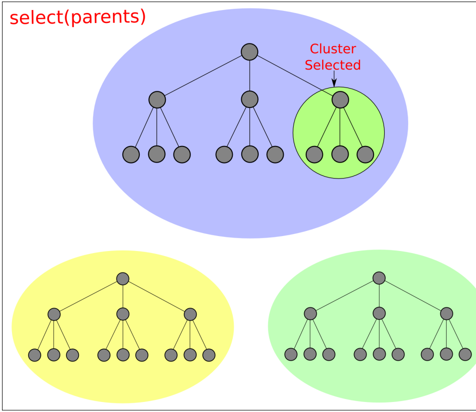


Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

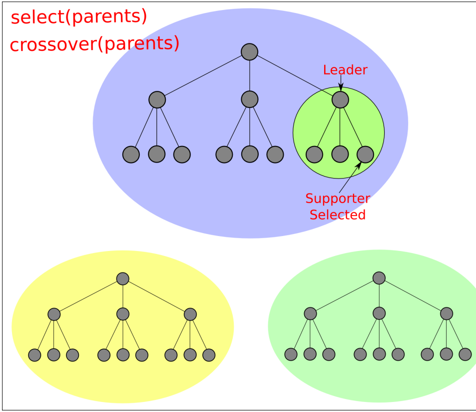


Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```



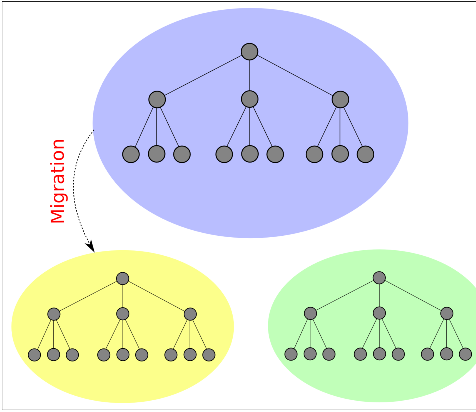
Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

i = 1



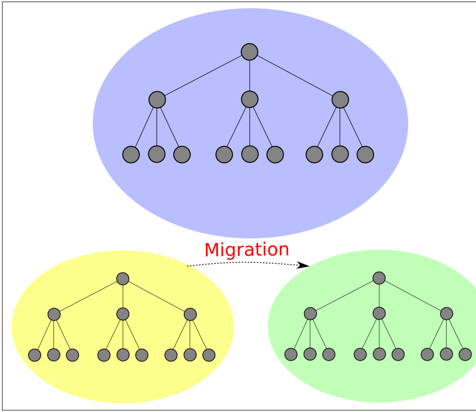
Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

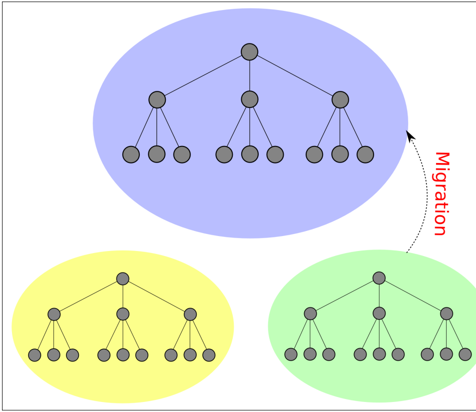
i = 2



Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

i = 3



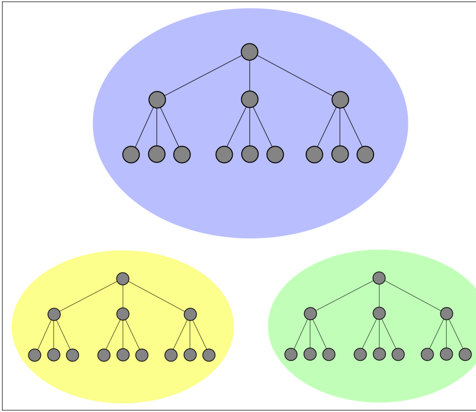
Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```

number of evaluations

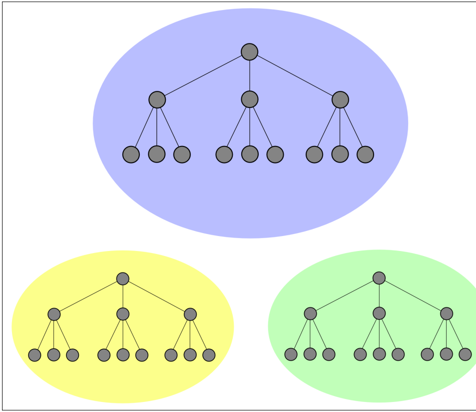


Methods

Multi-Population Genetic Algorithm

Algorithm 2: Multi-Population Genetic Algorithm.

```
begin
    repeat
        for i = 1 to numPop do
            for j = 1 to numIndividuals do
                initialize(pop(i).ind(j));
                evaluate(pop(i).ind(j));
            organize(pop(i));
            repeat
                for j = 1 to rateCross × numIndividuals do
                    select(parents);
                    child ← crossover(parents);
                    mutation(child);
                    evaluate(child);
                    add(child, pop(i));
                organize(pop(i));
            until converge(pop(i));
        for i = 1 to numPop do
            migrate(pop(i));
    until reach(stoppingCriterion);
    RouteLanding bestRoute ← getBestRoute(pop);
    return bestRoute;
```



Methods
Objective Function

$$minimize\ fitness = -C_{\phi_b} \cdot \sum_{i=1}^{|\phi_b|} (P(x_K \in Z^i_{\phi_b})) + C_{\phi_p} \cdot \sum_{i=1}^{|\phi_p|} (P(x_K \in Z^i_{\phi_p})) +$$

$$C_{\phi_n} \cdot max(0, 1 - \Delta - P\left(\bigwedge_{t=0}^{K} \bigwedge_{i=1}^{|\phi_n|} x_t \notin Z^i_{\phi_n}\right)) + \frac{1}{|\varepsilon_{max}|} \cdot \sum_{t=0}^{K} \|u_t\| \cdot |\varepsilon_t| +$$

$$shortestDist(\overline{x}_K, Z_{\phi_b}) + \begin{cases} C_{\phi_b} & , v_K - v_{min} > 0 \\ 0 & , \text{otherwise} \end{cases} + \begin{cases} C_{\phi_b} \cdot 2^{\frac{(K-T)}{10}} & , \psi = \psi_b \\ 0 & , \text{otherwise} \end{cases}$$

- Landing on ϕ_b (light blue)
- Landing on ϕ_p (pink)
- Landing and fly on ϕ_n (grey)
- Curves of the UAV (green)
- Distance to ϕ_b (orange)
- Time violation (yellow)
- Battery failure (blue)

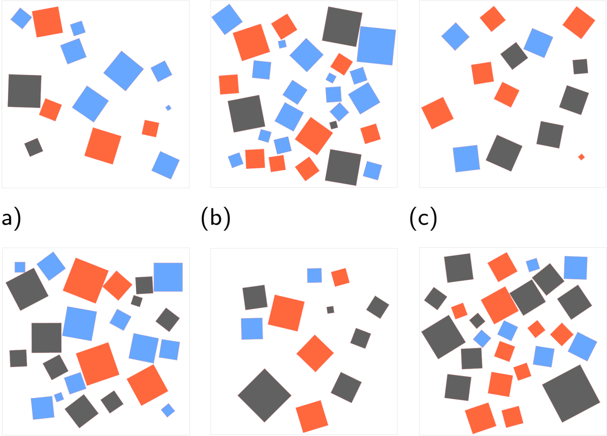
In this work, the following methods were used.

- **GH:** Greedy Heuristic
- **MPGA1(–GH):** Multi-Population Genetic Algorithm 1
    - Without greedy operator
- **MPGA2(+GH):** Multi-Population Genetic Algorithm 2
    - With greedy operator

Computational Results

Automatically Generated Maps

- Level of Difficulty
  1. M_E: a), b)
  2. M_N: c), d)
  3. M_H: e), f)
- Level of Coverage
  1. $C_{25\%}$: a), c), e)
  2. $C_{50\%}$: b), d), f)
- Legend Colors
  1. ϕ^b
  2. ϕ^p
  3. ϕ^n
  4. ϕ^r



(a) (b) (c) (d) (e) (f)

Computational Results

Parameters and Settings used in the Experiments

Model	Parameters	Value
Map	Dimension X [m]	1000
	Dimension Y [m]	1000
UAV	Initial Position (p_0^x, p_0^y) [m]	$(0; 0)$
	Initial Velocity (v_0) [m/s]	24
	Initial Angle (α_0) [°]	90
	Linear Velocity ($v_{min}; v_{max}$) [m/s]	[11; 30]
	Angular Variation ($\varepsilon_{min}; \varepsilon_{max}$) [°$/s$]	[-3; 3]
	Acceleration ($a_{min}; a_{max}$) [m/s^2]	[0; 2]
	Number of time steps to land (T) [s]	60
	Time Discretization (ΔT) [s]	1
	Probability of failure (Δ)	0.001
MPGA	Populations	3
	Individuals/Pop	13
	Individuals Total	39
	Mutation Rate	0.5
	Crossover Rate	0.75
	Stop Criterion	10000

Computational Results

Experiments: Result obtained for the GH, MPGA1(–GH) and MPGA2(+GH)

		GH			MPGA1(–GH)			MPGA2(+GH)		
Ψ	Instance	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.
ψ_m	M_E and $C_{25\%}$	79	21	0	81	19	0	90	10	0
	M_E and $C_{50\%}$	92	6	2	92	7	1	96	3	1
	M_N and $C_{25\%}$	58	39	3	60	39	1	71	28	1
	M_N and $C_{50\%}$	86	12	2	84	16	0	96	4	0
	M_H and $C_{25\%}$	30	52	18	36	64	0	40	60	0
	M_H and $C_{50\%}$	62	28	10	60	33	7	82	15	3
	Avg	67.8	26.3	5.8	68.8	29.7	1.5	79.2	20.00	*0.83*

Computational Results

Experiments: Result obtained for the GH, MPGA1(–GH) and MPGA2(+GH)

		GH			MPGA1(–GH)			MPGA2(+GH)		
Ψ	Instance	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.
ψ_b	M_E and $C_{25\%}$	99	0	1	100	0	0	100	0	0
	M_E and $C_{50\%}$	97	0	3	99	0	1	99	0	1
	M_N and $C_{25\%}$	93	3	4	94	5	1	99	0	1
	M_N and $C_{50\%}$	98	0	2	99	0	1	100	0	0
	M_H and $C_{25\%}$	67	5	28	73	27	0	94	6	0
	M_H and $C_{50\%}$	83	0	17	68	17	15	95	2	3
	Avg	89.5	1.3	9.2	88.8	8.2	3.0	97.8	1.3	*0.8*

Computational Results

Experiments: Result obtained for the GH, MPGA1(–GH) and MPGA2(+GH)

		GH			MPGA1(–GH)			MPGA2(+GH)		
Ψ	Instance	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.
ψ_{s^1}	M_E and $C_{25\%}$	81	8	11	90	8	2	91	7	2
	M_E and $C_{50\%}$	88	0	12	89	0	11	93	0	7
	M_N and $C_{25\%}$	68	16	16	76	18	6	86	8	6
	M_N and $C_{50\%}$	82	1	17	84	3	13	89	0	11
	M_H and $C_{25\%}$	41	23	36	49	46	5	67	28	5
	M_H and $C_{50\%}$	56	0	44	46	23	31	78	4	18
	Avg	69.3	8.0	22.7	72.3	16.3	11.3	84.0	7.8	*8.2*

Computational Results

Experiments: Result obtained for the GH, MPGA1(–GH) and MPGA2(+GH)

		GH			MPGA1(–GH)			MPGA2(+GH)		
Ψ	Instance	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.	ϕ_b	ϕ_r	Inf.
ψ_{s^2}	M_E and $C_{25\%}$	90	4	6	94	4	2	99	0	1
	M_E and $C_{50\%}$	90	0	10	95	1	4	95	1	4
	M_N and $C_{25\%}$	70	20	10	79	16	5	92	5	3
	M_N and $C_{50\%}$	87	1	12	83	8	9	94	0	6
	M_H and $C_{25\%}$	40	17	43	62	35	3	74	24	2
	M_H and $C_{50\%}$	61	3	36	57	13	30	76	4	20
	Avg	73.0	7.5	19.5	78.3	12.8	8.8	88.3	5.7	*6.0*
	Avg Final	74.9	10.8	14.3	77.1	16.7	6.2	87.3	8.7	*4.0*

Time (Sec)		
GH	MPGA1	MPGA2
0.07	1.017	0.874

Computational Results

Experiments: Example of Routes

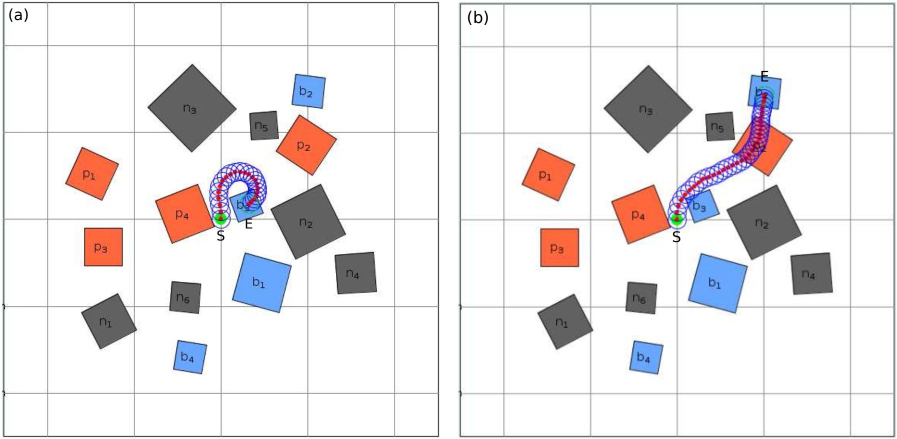


Figure 7: Routes determined by the planner MPGA2(+GH) in a map M_N with coverage $C_{25\%}$: (a) ψ_m. (b) ψ_b.

Computational Results

Experiments: Example of Routes

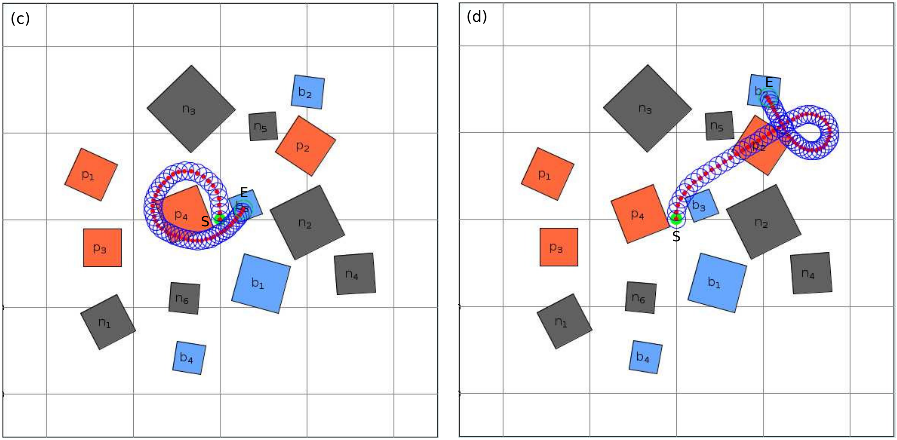


Figure 8: Routes determined by the planner MPGA2(+GH) in a map M_N with coverage $C_{25\%}$: (c) ψ_{s^1}. (d) ψ_{s^2}.

Computational Results

Experiments: Example of Routes

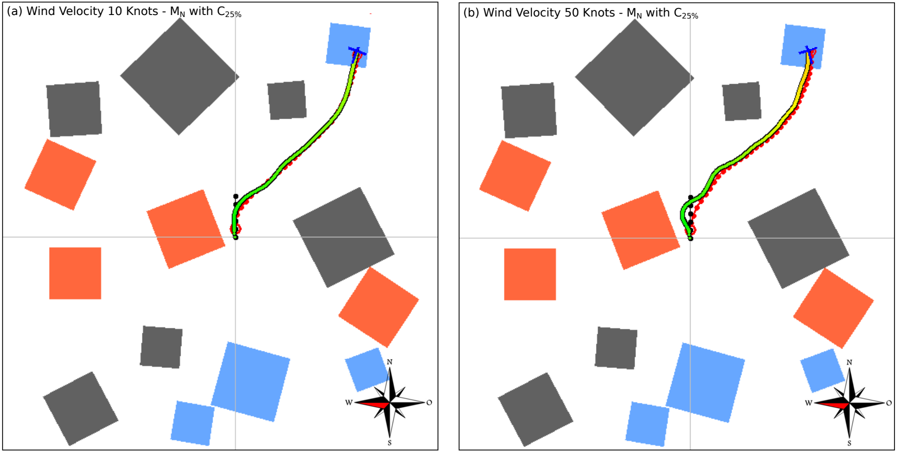


Figure 9: (a), (b) FG simulation with winds 10 and 50 knots. Wind direction: west.

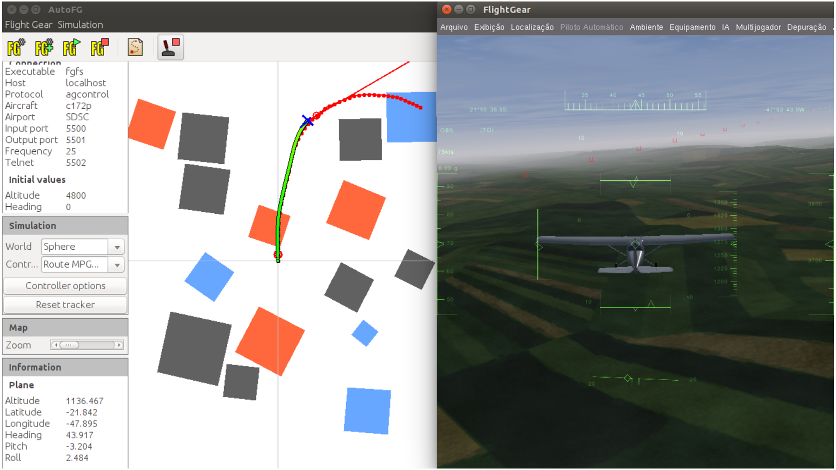


Figure 10: Video FlightGear Simulator.

Acknowledgements

Questions send email to:

jesimar.arantes@usp.br
marcio, claudio@icmc.usp.br
williams@mit.edu

Thank You!