

Project #2
Interprocess communication techniques under Linux
Due: December 12, 2025

Instructor: Dr. Hanna Bullata

Applying genetic algorithms for rescue operations on a collapsed building

We would like to develop a genetic algorithm-based application to optimize rescue operations under a single collapsed building. By rescue we mean delivering a small amount of nutrition and beverage for trapped people waiting for rubble removal operations to start. The application must generate efficient paths for rescue tiny robots in order to maximize survivor retrieval while minimizing operation time and risks.

We would like also to take advantage of the IPC techniques we've seen in class.

The project be explained as follows:

1. The project models a collapsed building as a 3D grid map where obstacles represent debris and accessible cells indicate potential survivor locations detected by sensors.
2. Assume heat and CO2 sensors will give indication on survivors under the debris.
3. Keep in mind we wish to optimize multi-robot path planning in a disaster environment. In addition, we wish to maximize coverage of survivor locations while minimizing travel time and collision risks.
4. We'll assume the environment will stay static during the rescue operations (e.g. real-time updates will not occur to change current situation post-aftershocks).
5. Keep in mind the following useful methodology:
 - **Chromosome Representation:** Paths encoded as sequences of grid coordinates.
 - **Fitness Function:** Must maximize coverage area and survivor reach while minimizing path length, risk exposure and time. Consider the following equation as an example:

$$f = w_1 \times survivors + w_2 \times coverage - w_3 \times length - w_4 \times risk$$

- **Selection:** Selection in genetic algorithms is the process of choosing individuals from the current population to act as parents for the next generation, favoring those with higher fitness values to mimic natural selection. Use tournament selection based on fitness to choose paths for reproduction.
- Apply other genetic operators such as **crossover** to exchange path segments and **mutation** to explore alternative routes. Use **elitism** to preserve top 10% solutions to avoid losing optima (make the 10% user-defined).
- Iteratively evolve the population over generations (evolution) until convergence criteria or maximum iterations are met.

What you should do

- Implement the above problem on your Linux machines using a multi-processing approach and take advantage of the IPC techniques we've seen in class.
- Generate a user-defined initial population (e.g. 50) random feasible paths using grid traversal (e.g. A^* seed).
- Evaluate fitness with collision detection.
- Evolve over a user-defined number of generations, stopping at stagnation or time limit (e.g. 100 - 300).
- The output must be optimized paths with survivor priorities, deployable to robots
- Be wise in the choices you make and be ready to convince me that you made the best choices :-)
- In order to avoid hard-coding input parameters, think of creating a text file that contains the different user-defined values a user might need to set. That will spare you from having to enter them over and over again. Give the file name as an argument to the application when run. That will spare you from having to change your code permanently and re-compile. Set default values in the code though in case the text file is missing when doing runs.
- Keep in mind you do not need to create children processes with each run. A better approach would be to keep the pool of created children processes for further computations.
- Test your program.
- Check that your program is bug-free. Use the `gdb` debugger in case you are having problems during writing the code (and most probably you will :-). In such a case, compile your code using the `-g` option of the `gcc`.
- The project lead is responsible to send the zipped folder that contains your source code and your executable before the deadline on behalf of the team. If the deadline is reached and you are still having problems with your code, just send it as is!