Comparing Cooperation and Competition Using Stakeholder Search

Olga E. Koldachenko

Department of Computer Science, Mount Royal University

COMP 5690: Senior Computer Science Project

Dr. Alan Fedoruk

April 24, 2022

Abstract

A stakeholder search framework was used to solve a randomly generated traveling salesman problem, where cooperative agents incorporated the data of other agents performing the search, while competitive agents iterated only on their own solutions. All agents ran the same genetic algorithm, with varying fitness functions to illustrate different goals. Cooperative agents performed significantly better than competitive agents in all cases. Another comparison was made for selfish agents, who share the solution best for their own fitness function, and selfless agents, who share the solution based on the chair's, with better results from selfish agents, which share more diverse solutions.

While cooperation gives better results overall, there's also a reduction in genetic diversity, which is compounded by selflessness and its accompanying reduction in the individuality of the solutions shared by the stakeholders. In the future, it would be worth exploring the benefits of a more diverse pool of stakeholders, including variations in the genetic algorithm being run and variations in the algorithm overall, instead replacing it with other forms of artificial intelligence like tree-based search or neural networks.

Comparing Cooperation and Competition Using Stakeholder Search

The goal of this project is to determine whether data sharing (cooperation) is beneficial within a multi-agent system (MAS). The framework used is *stakeholder search* (Fedoruk & Denzinger, 2006), where a *chair* agent distributes a problem to a set of *stakeholder* agents and each stakeholder has its own goals separate from the chair. In regular intervals (or *meetings*), the chair interrupts the stakeholders' search and requests their current progress, then re-distributes the best solutions given. A cooperative stakeholder incorporates these best solutions into its own search, while a competitive stakeholder ignores them. Another variable, *selflessness*, is included to determine what solution is shared by a given agent: a *selfless* agent shares the solution best for the chair, while a *selfish* agent shares the solution best for itself. During the final meeting, the chair selects the best solution based on its own goals.

The problem being solved is the *traveling salesman problem* (TSP), where each agent creates a graph from a given set of vertices. For a solution to be valid, it must be a *Hamiltonian cycle*, which is a circuit where every vertex except for the first and last (which must be the same vertex) must appear exactly once (Goodaire & Parmenter, 2018, p. 311). Every edge between the vertices is given a weight, and the goal is to generate a cycle of least weight. In this instance, two different weights are provided: distance, which is the Euclidean distance between the two points, and time, which is the time it would take to travel between the two points. This problem is notable as it's considered *NP-complete*: "it is an open question as to whether or not there exists an efficient (polynomial time) algorithm for its solution" (Goodaire & Parmenter, 2018, pp. 326-327).

The type of search being run by each stakeholder is a *genetic algorithm* (GA). Inspired by biology, a GA begins with a set of randomly generated solutions, called the *population*. Each population goes through a series of *generations* as randomly selected *parents* go through crossovers, which mix their genes much like sexual selection. To determine the viability of a solution, a fitness function is used, and the element of "survival of fittest" ensures that as time passes, the overall population contains better solutions (Russell & Norvig, 2010, pp. 126-129). For consistency, the parameters of the GA are the same between all agents.

Method

The project was programmed in Python 3.8.8 and the libraries used are NumPy and Pandas for data manipulation ("NumPy, n.d.; "Pandas", n.d.), Matplotlib for graphing ("Matplotlib", n.d.), and PyGAD for the implementation of the genetic algorithm ("PyGAD", n.d.).

V is the set of all vertices in a given traveling salesman problem of size *N*. Each vertex $v \in V$ is defined as $\{x, y, t\}$, where the x and y values are randomly generated between the values 1 and *N,* while *t* is a "terrain" value randomly generated between 0.5 and 2.0. The results of the generation are saved to a .csv file and can be found in the */problems* subdirectory of the example code.

When solving a TSP, a pair of lookup tables *(D, T)* is generated by the chair for ease of calculation, containing all the possible edges of the solution, where D is the distance table and T is the time table. The distance table is calculated as the Euclidean distance between all points, so $D = \{(v_i, v_j), \sqrt{[(x_j - x_i)^2 + (y_j - y_i)^2]} \mid x_i, y_i \in v_i \text{ and } x_j, y_j \in v_j\}$. The time table is created through a function $F\colon D \rightarrow T$ where $T = \{(v_i, v_j), d * t_i * t_j, \text{ where } d \in D, v_i, v_j \in d, \text{ and } t_i \in v_i\ t_j \in v_j\}$. This acts as an abstraction of difficulty of travel: a vertex with a low terrain value can imply paved roads and highways within the vicinity, while a high terrain value can imply off-road travel.

In addition, normalized versions of both of these tables are generated. Each individual agent has its own set of weights for distance and time, and the normalized tables are combined in order to create a single table $W = \{(x, y), z\}$ for determining a unique fitness for each agent of any solution, where z is the cost of traveling between the points x and y. This ensures that the agents have their own distinct goals and rank solutions differently.

A solution S is given as a sequence $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{N-1}, v_N), (v_N, v_1)\}$. The fitness function is defined as the inverse of the sum of all weights $\frac{1}{\Sigma z}$ where $z \in w$ and $s \in w$. Therefore, a smaller weight will result in a larger fitness as required by the GA.

Each test is determined by two configuration files. The */server_configs* subdirectory contains the setup for several tests. This manipulates the size of the problem, the number of meetings and the waiting time between them, the number of duplicates of each stakeholder, and the number of top solutions to share at every meeting. The configurations that were run for the purposes of the experiment are coop.csv for all-cooperative stakeholders, comp.csv for all-competitive stakeholders, selfless.csv for all-selfless stakeholders, and selfish.csv for all-selfish stakeholders.

The */client_configs* subdirectory, which contains files of the same names, holds the client/stakeholder information for each configuration. This includes their distance and time weights as well as the flags determining whether or not each client in the setup is cooperative/competitive and selfish/selfless.

The data is logged in a timestamped subdirectory in the */logs* directory. Each stakeholder has a corresponding text file which records the state of its GA at every generation. The file *Server.csv* stores the fitness submitted by each stakeholder at each meeting. As well, the top solution is graphed and saved in *Solution.png*.

All tests were run using a TSP of size 100, found in */problems/tsp100.csv*. As well, each test was run for 60 meetings at 30 second intervals. The population multiplier was left at 1 to keep all stakeholders unique, and the number of solutions shared by the chair at each meeting was 5. The results shown here can be found in the */results* subdirectory.

The chair has a balanced priority, it has a weight of 0.5 for both time and distance. A stakeholder with a priority of time has a weight of 1.0 in time, while a stakeholder with a priority of distance has a weight of 1.0 in distance.

To get averages and reduce the random nature of the GA, each test was run 3 times.

Results

All-Competitive

Since every stakeholder in this test does not incorporate the results of others, this essentially is several individual competing searches, acting as a baseline for the effectiveness of the GA.

Table 1

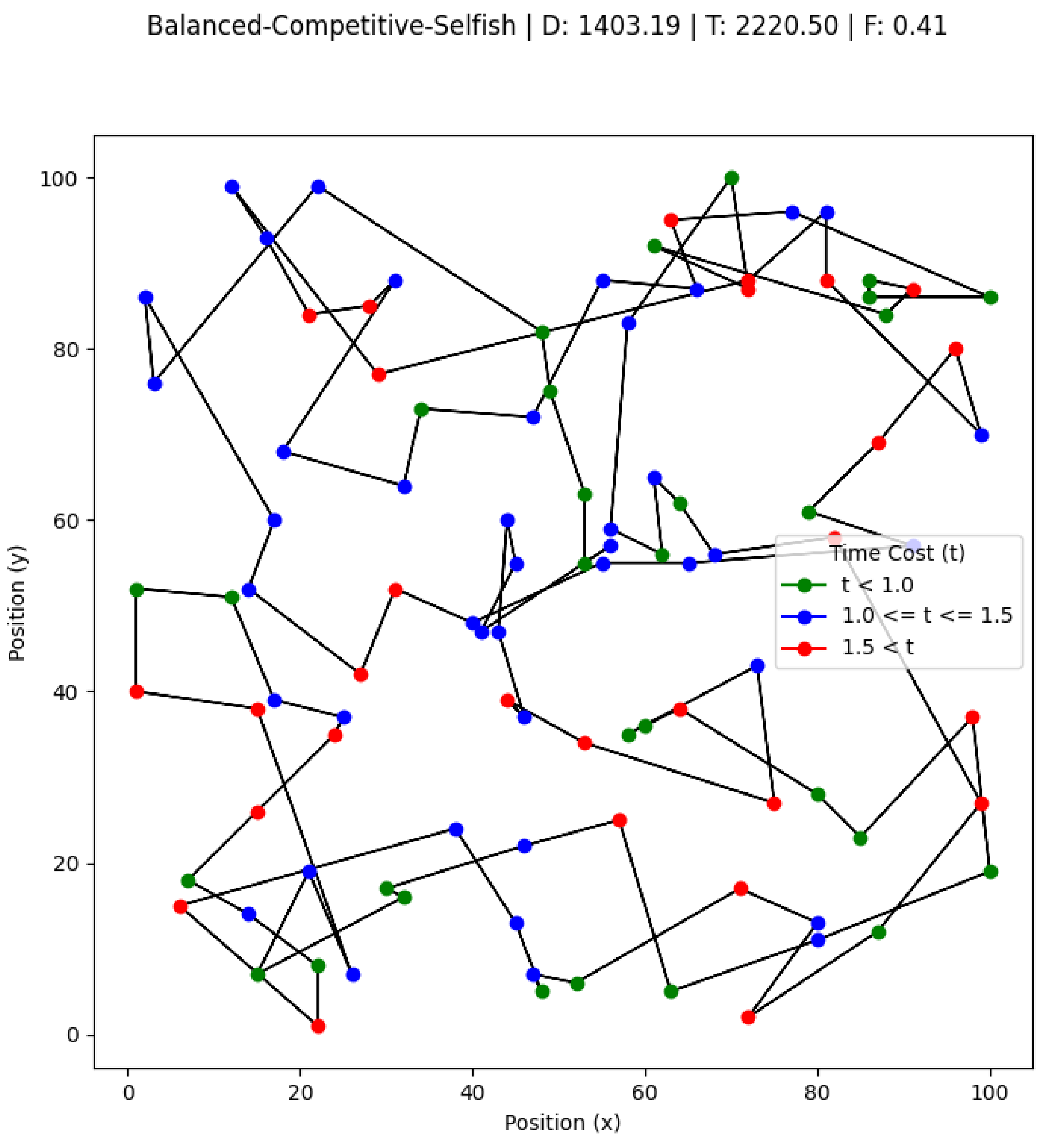
Results of Competitive Test

Priority	Cooperation	Selflessness	Fitness		
			Test 1	Test 2	Test 3
Distance	Competitive	Selfless	0.3502	0.3606	**0.3844**
Distance	Competitive	Selfish	0.3854	0.3748	0.3628
Balanced	Competitive	Selfless	0.3619	**0.4016**	0.3504
Balanced	Competitive	Selfish	**0.4051**	0.3776	0.3792
Time	Competitive	Selfless	0.2811	0.3134	0.3409
Time	Competitive	Selfish	0.3077	0.3494	0.3195

Note. Best result in bold, repeated result in italics.

Figure 1

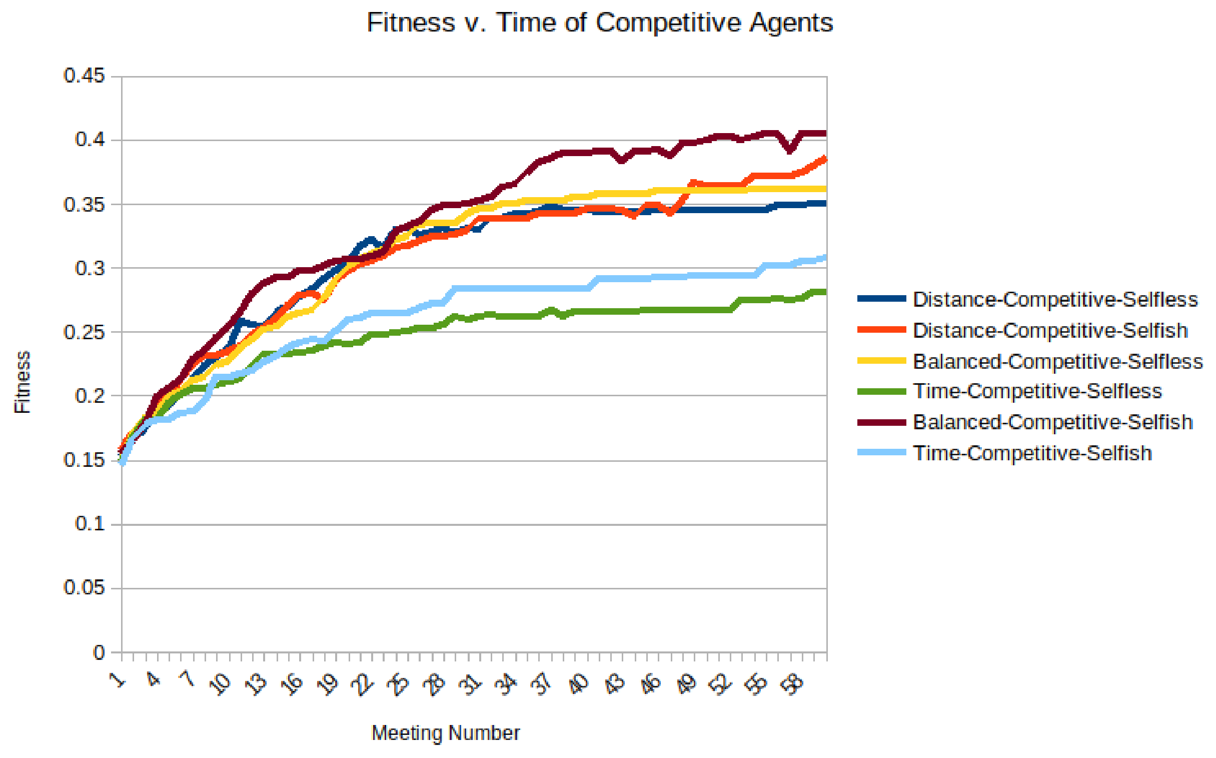
Best Solution From Competitive Test 1



Note that the colour of each vertex in the graph represents its *t* value: red means the node is expensive to reach while green means it's inexpensive. This could explain some unintuitive decisions between points when considering Euclidean distances. D is the numerical distance, T is the numbercal time, while F is the fitness as determined by the chair.

Figure 2

Fitness Graph For Competitive Test 1



All-Cooperative

This test ensures that every stakeholder is cooperative, using others' results.

Table 2

Results of Cooperative Test

Priority	Cooperation	Selflessness	Fitness		
			Test 1	Test 2	Test 3
Distance	Cooperative	Selfless	0.4980	*0.4981*	0.4329
Distance	Cooperative	Selfish	**0.4985**	0.4849	0.4222
Balanced	Cooperative	Selfless	*0.4981*	*0.4981*	***0.4342***
Balanced	Cooperative	Selfish	0.4884	**0.5020**	***0.4342***

Table 2 (continued).

Time	Cooperative	Selfless	*0.4981*	*0.4981*	0.4273
Time	Cooperative	Selfish	0.4870	0.4976	0.4248

Note. Best result in bold, repeated result in italics.

Figure 3

Best Solution From Cooperative Test 1

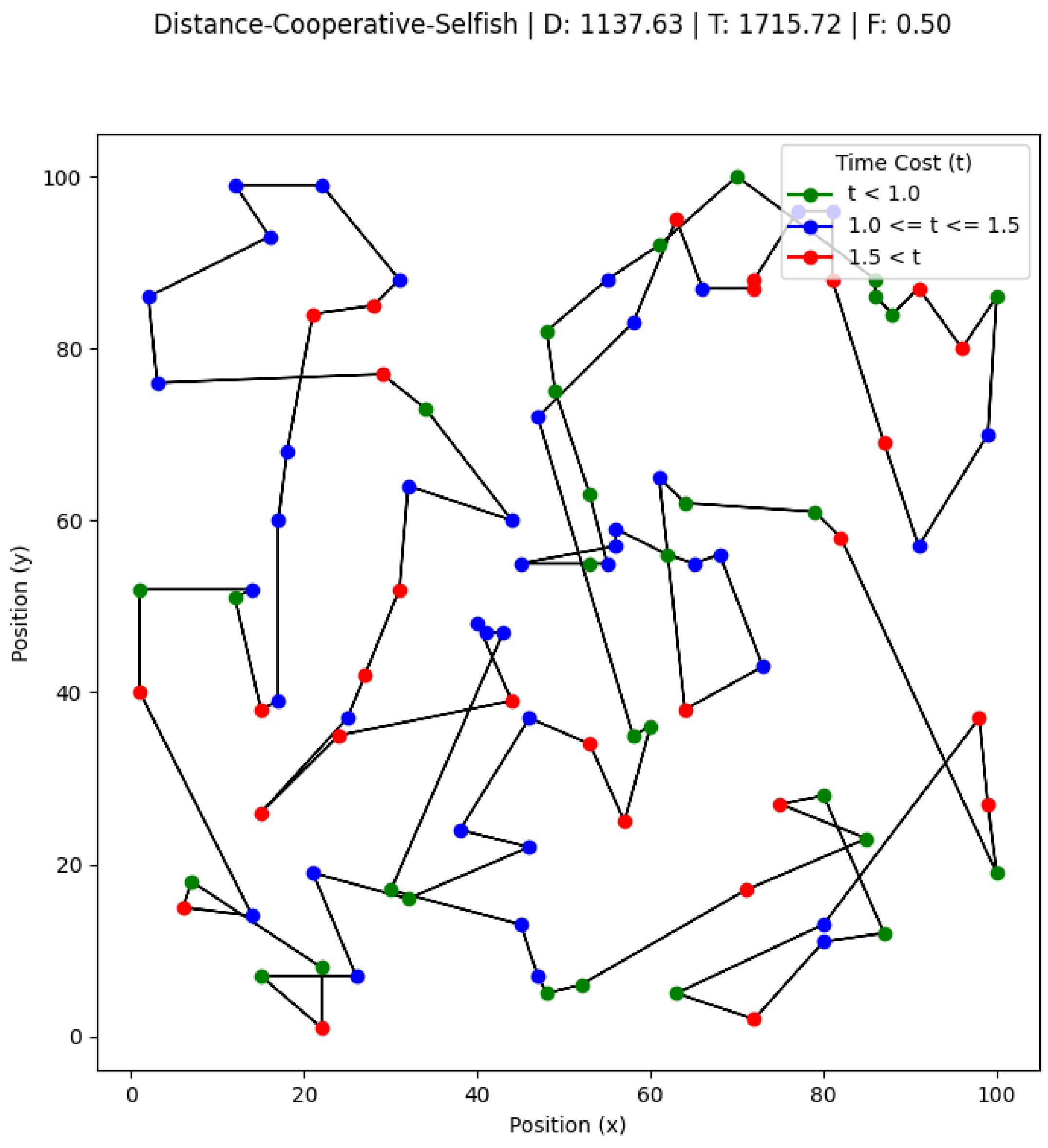
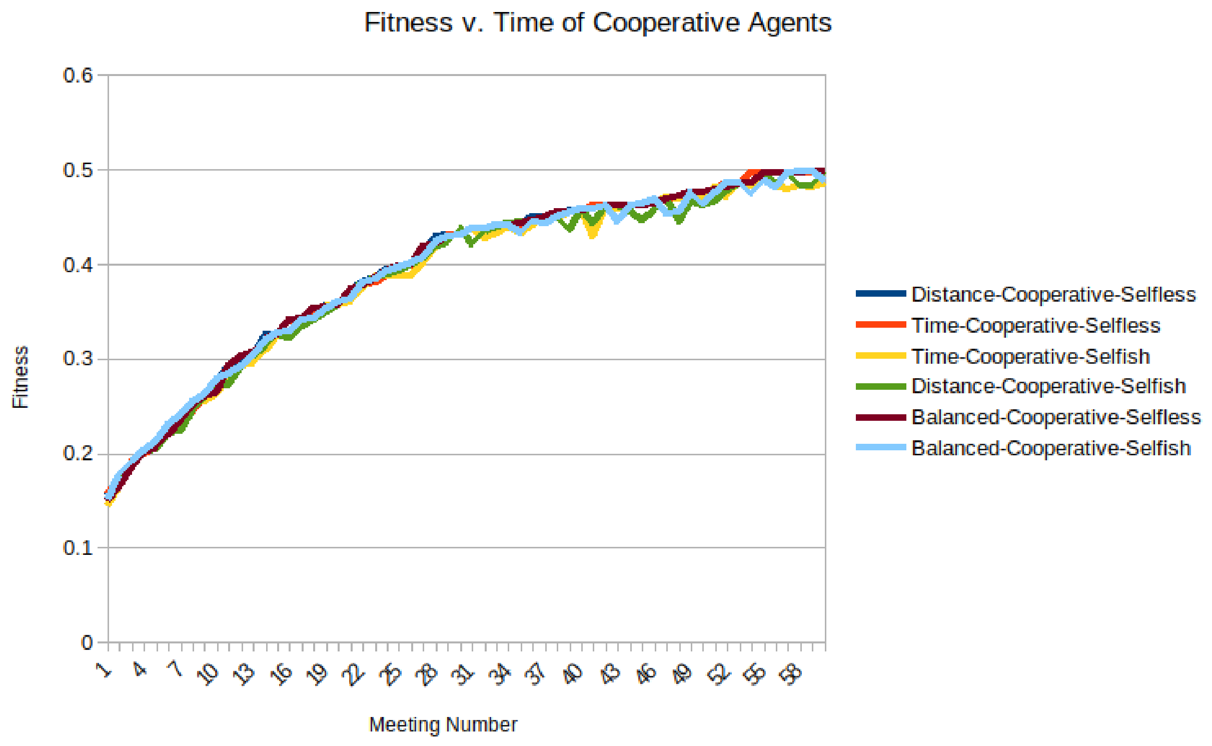


Figure 4

Fitness Graph For Cooperative Test 1



All-Selfish

At this test, cooperative and competitive agents are mixed, and now the sharing aspect of the agent's strategy is looked at in contrast to the taking aspect of it. Selfish stakeholders share the solution that is best for themselves, rather than using the chair's weights.

Table 3

Results of Selfish Test

Priority	Cooperation	Selflessness	Fitness		
			Test 1	Test 2	Test 3
Distance	Competitive	Selfish	0.3856	0.4177	0.3967
Distance	Cooperative	Selfish	**0.4762**	0.4582	**0.4557**

Table 3 (continued).

Balanced	Competitive	Selfish	0.3734	0.3455	0.3158
Balanced	Cooperative	Selfish	0.4759	0.4772	0.4436
Time	Competitive	Selfish	0.3263	0.3254	0.3815
Time	Cooperative	Selfish	0.4741	**0.4800**	0.4537

Note. Best result in bold, repeated result in italics.

Figure 5

Best Solution From Selfish Test 1

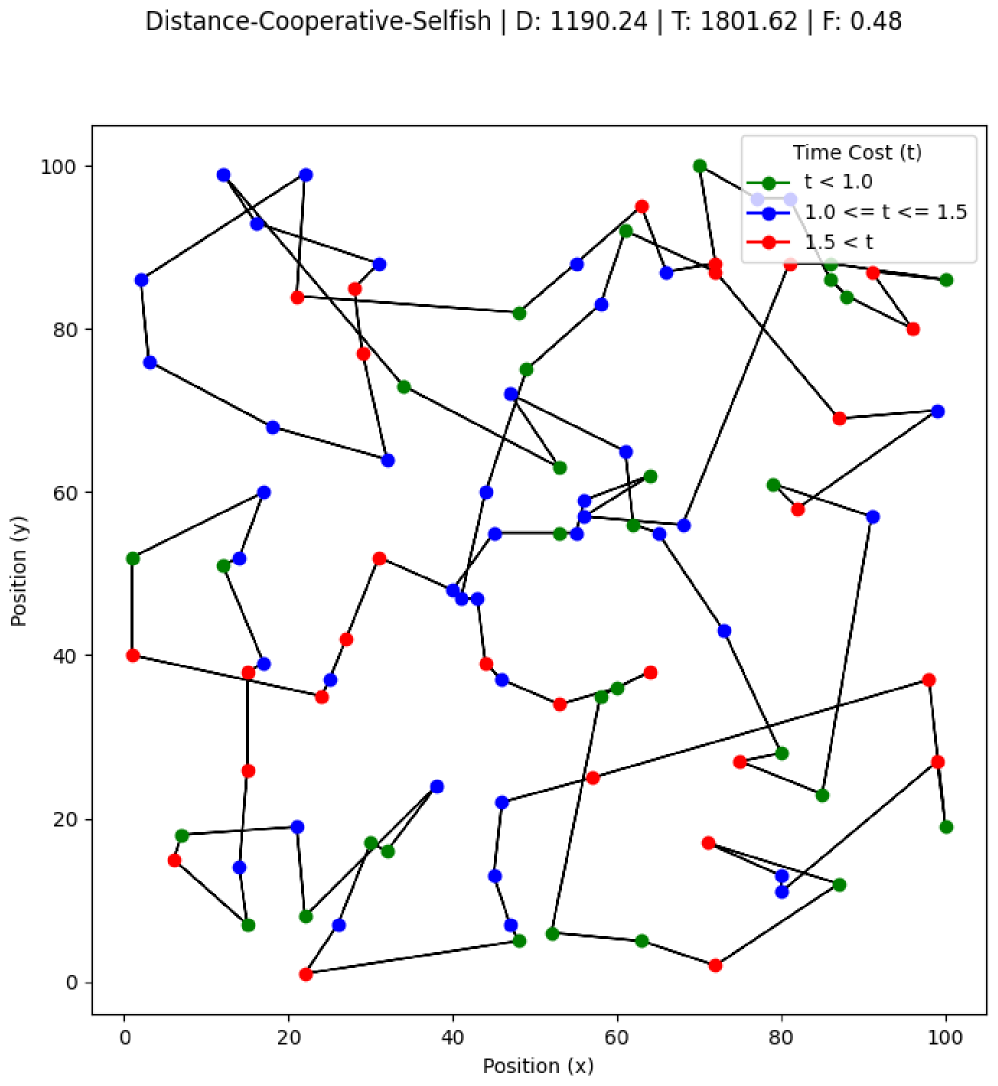
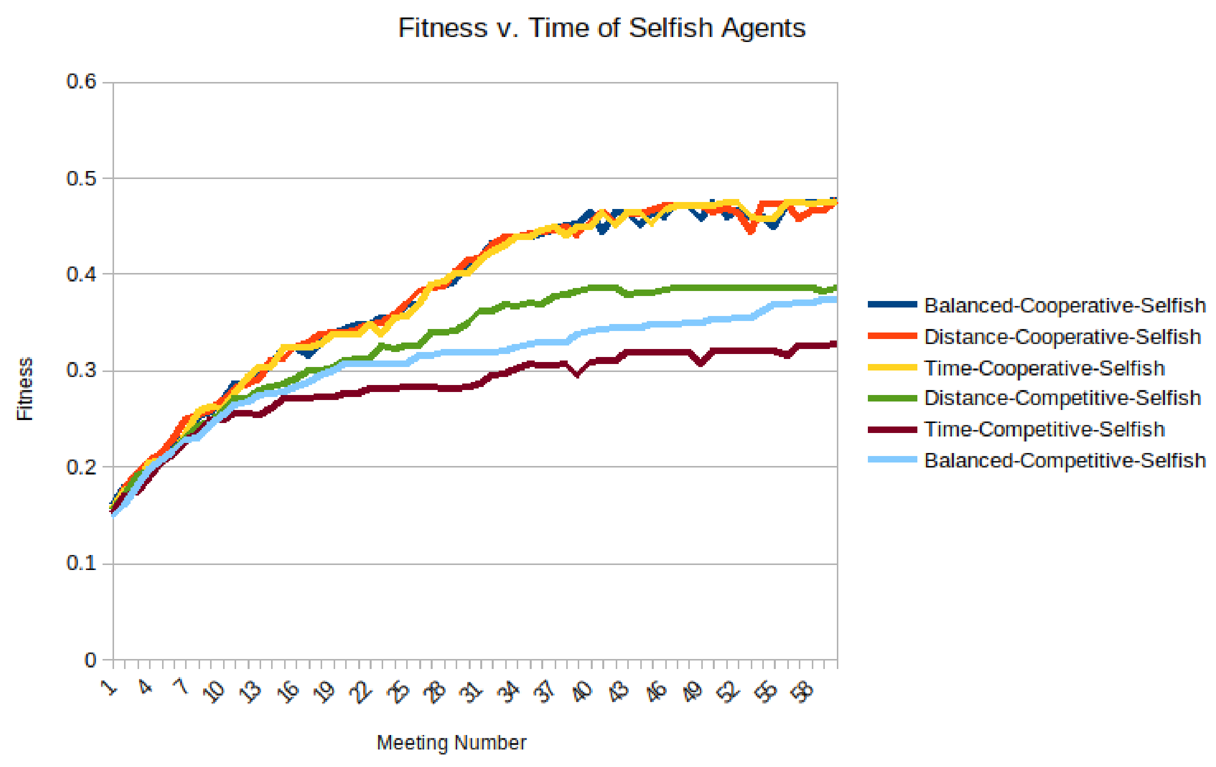


Figure 6

Fitness Graph For Selfish Test 1



All-Selfless

To contrast the all-selfish test, the stakeholders in this test only share the result best for the chair, rather than for themselves.

Table 4

Results of Selfless Test

Priority	Cooperation	Selflessness	Fitness		
			Test 1	Test 2	Test 3
Distance	Competitive	Selfless	0.3616	0.3936	0.3785
Distance	Cooperative	Selfless	***0.4522***	*0.4223*	0.4695
Balanced	Competitive	Selfless	0.3485	0.3861	0.3589

Table 4 (continued).

Balanced	Cooperative	Selfless	***0.4522***	*0.4223*	**0.4703**
Time	Competitive	Selfless	0.2963	0.3174	0.3193
Time	Cooperative	Selfless	***0.4522***	**0.4229**	0.4680

Note. Best result in bold, repeated result in italics.

Figure 7

Best Solution From Selfless Test 1

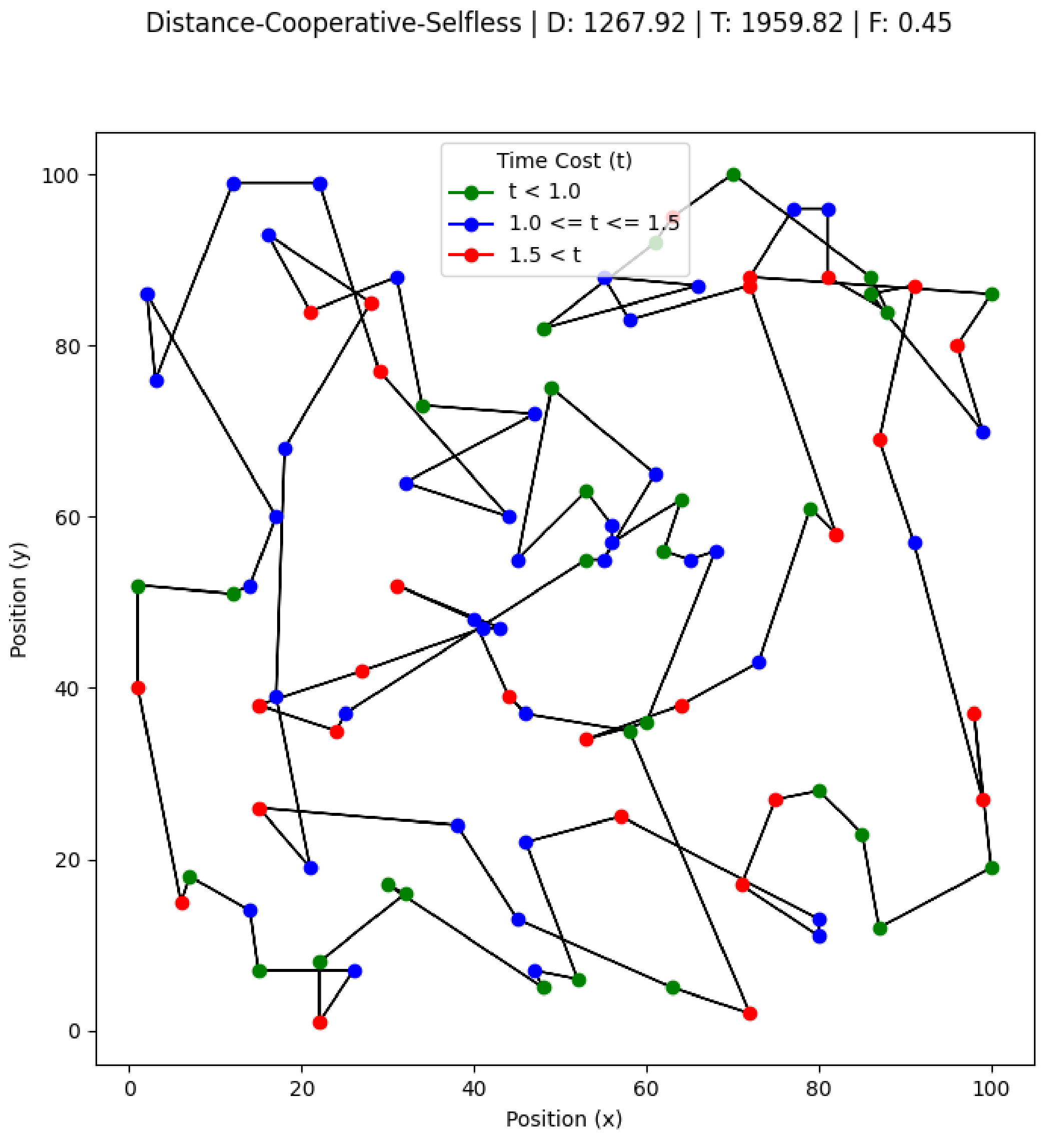
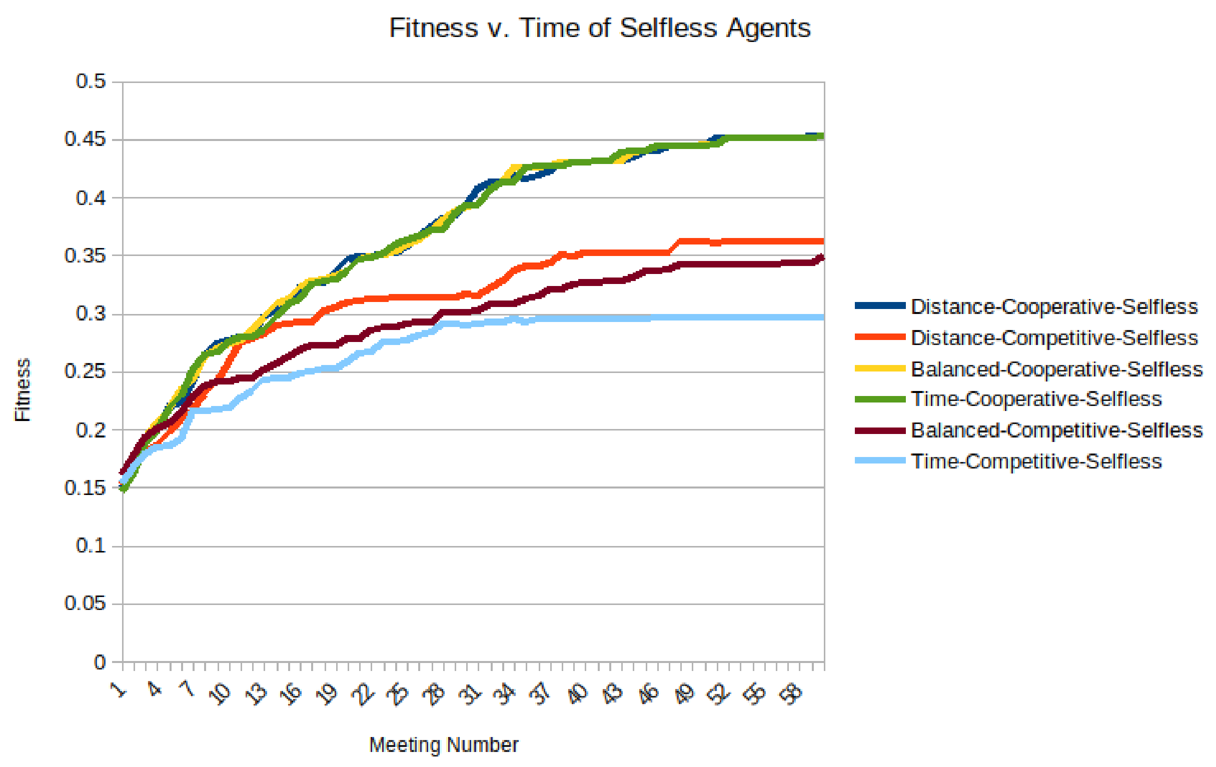


Figure 8

Fitness Graph For Selfless Test 1



Conclusion

The average fitness of the best results of the all-competitive test is 0.3970 (Figure 1), while the average fitness of the all-cooperative test is 0.4782 (Figure 2), resulting in a difference in fitness of 0.0812. The benefits of cooperation over competition are clear: while competitive stakeholders are as effective as the algorithm they're running, cooperative stakeholders "prop each other up" so that the weakest of them becomes the baseline to improve upon.

A notable side-effect of cooperation, however, is homogenization of the solutions. This could be a side-effect of the genetic algorithm, as "it is often the case that the population is quite diverse early on in the process, so crossover … frequently takes large steps … early in the search process and smaller steps later on when most individuals are quite similar" (Russell & Norvig, 2010, p. 128). Once cooperative agents are introduced into the system, some agents begin to resemble each other very strongly and share the same results. One thing worth considering in the future is the value of having a more varied gene pool. For the two mixed tests, however, the selfless agents resulted in an average of 0.4484 and the selfish agents resulted in an average of 0.4706. Neither of these results are better than the all-cooperative test, so the benefit of competitive agents is dubious.

The difference between selfishness and selflessness is less clear. The selfish test performed better on average, but the difference created is less pronounced than the difference between cooperation and competition (0.0222 compared to 0.0812). It did however result in a higher diversity of solutions. In the selfless test, while the cooperative stakeholders outperformed the competitive ones, they also had a higher tendency to resemble each other than the selfish ones. It's possible that the selfish solutions, being decided by more varied fitness functions (individual to each stakeholder), helps promote diversity within the gene pool to counteract the possibility of homogeneity in the cooperative agents.

Thirty minutes may not have been enough to fully realize the potential of the individual GAs, and some extra time per test may result in better fitness graphs, with a longer horizontal tail. Another potential change would be to run more tests to even out the random nature of the GA, and introduce some more types of artificial intelligence.

References

Fedoruk, A., & Denzinger, J. (2006). A General Framework for Multi-agent Search with Individual and Global Goals: Stakeholder Search. *International Transactions on Systems Science and Applications (ITSSA), 1(4), 357-362*. **doi: 10.1.1.381.2890**

Goodaire, E.G., & Parmenter, M.M. (2018). *Discrete Mathematics with Graph Theory* (3rd ed.) Pearson Education.

NumPy. (n.d.) NumPy. **https://numpy.org/**

Pandas. (n.d.) Pandas. **https://pandas.pydata.org/**

PyGAD - Python Genetic Algorithm! (n.d) PyGAD. **https://pygad.readthedocs.io/en/latest/index.html**

Matplotlib: Visualization with Python. (n.d.) Matplotlib. **https://matplotlib.org/**

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.) Pearson Education.