NTNU – Trondheim
Norwegian University of
Science and Technology

Investigating the value of communication in spacecraft swarms for asteroid prospecting operations

Erik Stokken Samuelsson
Magnus Hertzberg Ulstein

Master of Science in Computer Science
Submission date: July 2015
Supervisor: Axel Tidemann, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Communication between spacecraft in deep space is difficult and costly. On the other hand, autonomous missions may behave suboptimally when starved for information. We investigate the balance between communication and efficiency in planned asteroid exploration missions involving multiple cooperating spacecraft.

We produce a map over all available transfer windows in a 20 year period from a given initial orbit within the asteroid belt. A swarm of spacecraft is tasked with exploring the asteroid belt via these transfer opportunities, using an evolutionary algorithm to plan its routes. The amount of information available to each spacecraft is varied between no information, occasional aggregated updates and complete information about the plans of the other spacecraft.

Comparing the performance of swarms with different amounts of information, we examine the efficiency loss, in terms of scientific return, from limiting their ability to coordinate and conclude that coordination is not necessary under the examined conditions. The sheer size of the asteroid belt, and the wide variety of options means that autonomous spacecraft are unlikely to decide to explore the exact same asteroids. We find that it is not necessary to spend any significant amount of resources on maintaining communication within the swarm. At least, unless the number of target asteroids is significantly reduced compared to the full asteroid belt or a larger number of spacecraft is involved.

Sammendrag

Kommunikasjon mellom romfartøy er krevende, men det er en mulighet for at autonome oppdrag yter dårligere hvis kommunikasjonsnivået begrenses. Vi undersøker sammenhengen mellom ytelse og kommunikasjon for planlagte asteroideutforskningsoppdrag som involverer flere samarbeidende fartøy.

Vi produserer et kart over alle reisemuligheter mellom asteroidene i hovedbeltet i en 20 års periode, med utgangspunkt i en gitt startbane. En sverm av romfartøy har som oppdrag å utforske asteroidebeltet via reisemulighetene fra kartet. Kunstig evolusjon brukes til å planlegge fartøyenes ruter, med varierende tilgang til informasjon om hva resten av svermen foretar seg. Alternativene vi skal se på er: ingen informasjon, regelmessige oppdateringer med begrenset informasjon og øyeblikkelig full informasjon.

Ytelsesforskjellen mellom kommunikasjonsnivåene undersøkes med tanke på vitenskapelig gevinst. Konklusjonen er at delt informasjon ikke er nødvendig for de undersøkte forholdene. På grunn av den enorme størrelsen til asteroidebeltet og de mange valgmulighetene er det lite sannsynlig at ukoordinerte fartøy velger de samme rutene. Det ser ut som det ikke er viktig å bruke store mengder ressurser på å opprettholde kommunikasjon internt i svermen. Dette gjelder med mindre langt færre asteroider anses som gode mål eller langt flere romfartøy er involvert.

Preface

This thesis is made as the completion of our master's education in Computer Science for the Department of Computer and Information Science at the Norwegian University of Science and Technology.

Acknowledgements

First of all, we would like to thank our multiple advisers in this project. Pauline Haddow during the specialisation project, Keith Downing in the final preparations before the thesis project itself, and especially Axel Tidemann who advised us throughout the writing of the thesis. We would also like to thank the people who assisted Pauline in advising us while she was in Australia, Jean-Marc Montanier and Christian Skjetne. We would also like to thank our friends and family for proof reading and suggestions. And finally, we would like to thank Phil Palmer and Andrew Carrol for answering our questions about their stigmergy based evolutionary planner.

Contents

Figures

Tables

1 Introduction

Deep space communication is costly and difficult, but the cooperation of autonomous spacecraft may suffer without it. The exact trade off between communication cost and efficiency loss is important for future space missions involving multiple autonomous spacecraft. We seek to investigate this trade off.

1.1 Motivation

Space agencies are currently investigating a new class of multi-spacecraft missions.[1,2] A typical target of these missions is the main asteroid belt.[3,4] Given the huge variety of asteroids and the limited lifespan of all spacecraft, a single probe would only be able to explore a scant handful of asteroids.[5] In order to answer the many open questions about them, visiting dozens or even hundreds may prove necessary.[6] This is far beyond the capacity of a single spacecraft.

Traditionally, each spacecraft would be controlled individually from Mission Control on Earth. In this case, the sheer number of spacecraft would require a tremendous amount of manpower. Even simple satellite missions today have three or four mission controllers for every satellite in orbit.[2] The obvious solution is to make the vehicles more autonomous, but this leads to another problem; one of coordination.

Communication is central to good cooperation, also for autonomous agents. Too little, and each agent pursues its own goals independently of the rest. Too much, and important time is wasted communicating unnecessary information. This is not a trivial problem and compromises must be painfully worked out.[7]

For spacecraft, communication costs more than just time. To receive and transmit data without using too much power, spacecraft must use a directional antenna. Such an antenna needs to be aimed, meaning the receiver and transmitter must be pointed towards each other.[8] Consequently, both parties must know in advance when communication will happen and the direction of the target. The antenna is often rigidly mounted, meaning the entire spacecraft must be rotated to aim it.[4] Rotating the spacecraft requires power for flywheels or propellant for thrusters. The fixed orientation of the spacecraft means that, during communication, rigidly mounted solar panels are not angled to receive the maximum amount of light. Similarly, the propulsion system has a very limited vector it can thrust in, making manoeuvring all but impossible. To make matters worse, the limited power and vast distances involved limits the bit rate to hundreds of bits/s at most.[4]

In the scarcely populated asteroid belt, target asteroids can be millions of kilometres apart. Spacecraft spread out significantly as they explore outwards through the belt and communication gets more and more difficult. But how much is the performance of a swarm of spacecraft reduced, if at all, when their ability to communicate lessens? Limiting the communication needs of a spacecraft frees up resources for other purposes and allows it to navigate more freely. With less need for resources, a lighter and cheaper platform can be used. However, limiting the amount of communication makes cooperation more difficult, and a compromise must be struck. But before we can attempt to find a reasonable compromise we need to know what the spacecraft will be doing.

The largest open question about the asteroid belt is the composition of the asteroids themselves.[9] While Earth based spectrographic analyses has been able to gleam some understanding of their chemical structure, the macroscopic build-up is hard to determine via telescope. Asteroids could be loose piles of dust and rocks held together by mutual gravitational attraction. Alternatively, they could be solid objects fused together by repeated collisions; its surface covered by more recent debris and dust. Analysis is further complicated by there being at least a dozen classes with very different chemical make up.[10] It's fully possible that some asteroid classes are “piles of rubble”

while others are “dusty solids.”[11]

The question of asteroid composition is not merely academic. Should it become necessary to deflect an asteroid that would otherwise impact Earth, it makes a big difference whether it is a dusty solid or a rubble pile. A dusty solid has the structural integrity required to allow us to mount rocket engines on it, or use a nuclear warhead to vaporise part of it as an impromptu rocket engine.[12] Attempting to deflect a rubble pile with these methods would only result in it scattering, so they require entirely different strategies than dusty solids.

Similarly, the fledgling asteroid mining industry is very interested in asteroid composition. Mining techniques vary depending on the material in question. Solid asteroids are easier to attach spacecraft to, while loose regolith can be easily scooped up with light machinery. Furthermore, information about the density of an asteroid tells whether it might have an icy core, which would make it a valuable source of hydrogen propellant. Greater understanding of the asteroids simplifies asteroid mining; which in turn, through cheap in-space access to propellant, reduces the cost of future space exploration.[13]

Unfortunately, the asteroids have not seen as much study as the Moon or Mars. Over the past 50 years, we have sent probes to every planet, but less than a dozen asteroids have ever even received a flyby. While classification schemes differ, most divide asteroids into 4-24 categories based on spectrography alone.[10] The greatest achievement to date has been the Japanese Hayabusa probe taking samples of surface of 25143 Itokawa and managing to return them to earth despite the mission's constant technical issues.[14]

Otherwise, progress has been limited, although there are some interesting ongoing projects. Hayabusa 2 is currently headed for 1999 JU_3; having hopefully ironed out the technical issues that plagued its predecessor.[15] NASA intends to launch OSIRIS-REx in 2016[16], and the Dawn mission is currently orbiting 1 Ceres (the first, and largest, asteroid ever discovered)[17]. Two commercial companies have also announced plans for asteroid prospecting: Deep Space Industries and Planetary Resources.

These projects have only a few targets at most, leaving a lot to be explored. A more thorough survey is needed to fully understand the diversity and composition of asteroids in the solar system. Perhaps the most notable such project, NASA's PAM/ANTS mission,[3] was cancelled years ago. It would have involved a thousand 1kg spacecraft, making extensive use of future nanotechnology. The European Space Agency (ESA) has a far more conservative project called the Asteroid Population Investigation & Exploration Swarm (APIES) in its early stages of planning.[4]

APIES, which is described in detail in section 2.2, consists of 20 spacecraft; 1 HIVE and 19 BEEs. The BEEs assume a formation around the HIVE and intercept passing asteroids. During a flyby, they will capture as much data as possible while still remaining at a distance. The mission expenses would be unacceptably high if these spacecraft are all to be remote controlled without extensive automation.

1.2 Research Goals

The traditional strategy of remote controlling all spacecraft from Earth would not be feasible for APIES. ESA plans to use swarm intelligence, but has not yet decided on any details. We hope to contribute in their decision making by examining the value of communication for a simplified model of the mission. Using a simple swarm based technique, we compare how the BEEs perform with varying amounts of shared information. Hopefully, this will help ESA find the proper balance between communication overhead and duplication of effort.

1.3 Research Questions

We use a modified version of the APIES mission as explained in chapter 4. Our BEEs do not perform flybys of passing asteroids, but seek out and orbit asteroids on their own initiative.

We will investigate how well a swarm of spacecraft operates with varying amounts of communication. The spacecraft will use an evolutionary algorithm to navigate a simplified model of the asteroid belt. Different fitness metrics are used depending on the amount of information available to the individual spacecraft.

To evaluate our methods, we will use a Monte Carlo Simulation, described in section 3.6, to generate a large set of random solutions. We can compare the performance of our methods to this statistical baseline. This both allows us to see how much better than random our solutions are and how close to optimum we can get.

The first test is the maximum attainable performance if the spacecraft are guided from Earth. The planning is done with full access to all information known to the Earth-based Mission Controllers. To simplify the problem, it is assumed that everything goes according to plan and no replanning is necessary during the mission. As such, our first research question is:

RQ1 What is the maximum efficiency we are able to achieve when all spacecraft plan collectively before the start of the mission and everything goes according to plan?

This is equivalent to giving orders to every spacecraft individually from an Earth based autonomous system. Although this is certainly possible, it may be desirable to limit the use of the Earth based Deep Space Network transmitters if the loss in efficiency is not too high. Tripp and Palmer's stigmergy method, described in section 3.5, is an established evolutionary method for reducing the communication requirements of spacecraft swarms. In order to quantify this efficiency loss, we formulate our second research question:

RQ2 Is there a significant loss in efficiency when stigmergy mechanisms are used to reduce the communication requirements of the swarm?

Given the sparsity of the asteroid belt, the loss in efficiency if there is no coordination at all may not be insurmountable. Having a collection of individual agents would not be as much of a problem if their initial dispersion goes well. Heterogeneity would help them avoid independently following the same plan, but:

RQ3 Is heterogeneity enough to ensure reasonable efficiency even with no communication?

The three methods will be compared to one another and to the Monte Carlo baseline. We will vary the rate of stigmergy updates and the level of heterogeneity to see how this affects the results. All results will be placed in a two-dimensional graph showing their efficiency in the solution space. The graph is a histogram of the utility versus percentage of solutions with that utility.

1.3.1 Hypotheses

We expect the distribution of the solution space to favour low efficiency solutions, displaying an inverse relationship between scientific return and number of solutions with that level of scientific return, as seen in Figure 1. The vast majority of randomly generated routes should be too propellant intensive for the spacecraft and so visit few and low value targets.

Our best case scenario with full communication (RQ1) is not expected to reach the maximum efficiency solutions found by Monte Carlo. Our algorithms will always have some inefficiency that Monte Carlo Simulation will avoid because of the low probability that none of the randomly generated solutions will be extremely efficient. However, we still expect our minimum expected utility to be well within the upper five percentiles of the solution space, making it better than at least

95% of the solution space.

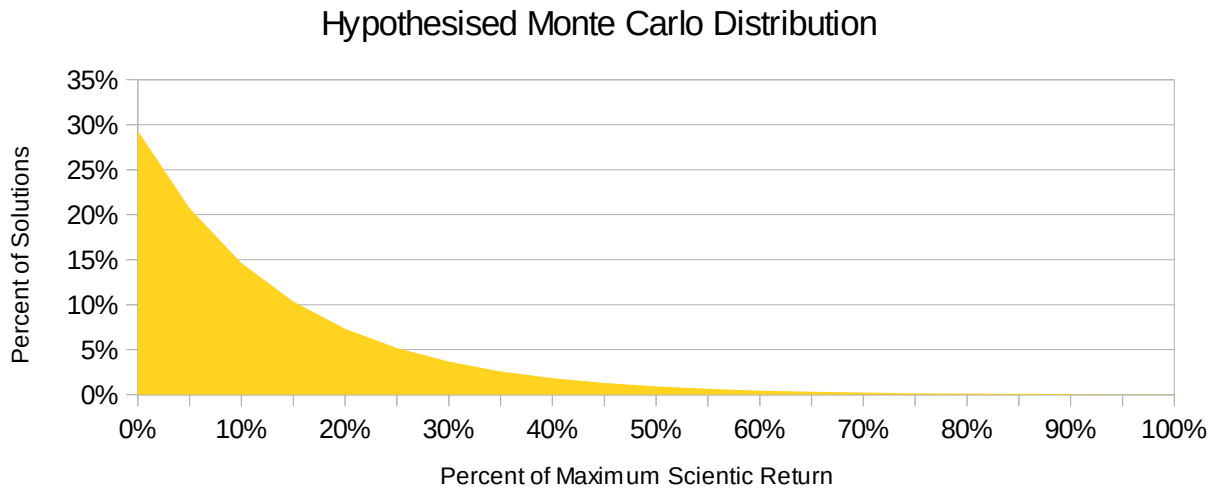

Figure 1: Shows the hypothesised distribution of routes found during Monte Carlo Simulation according to percentage of the highest scientific return found. We hypothesise that the number of solutions in each 10% interval will be a fraction of the solutions in the previous. Only a small minority is expected to achieve a high scientific return.

Stigmergy (RQ2) should display a wide range of utilities depending on the level of heterogeneity and how often updates are sent out. However, we still expect to find parameters for which it performs at a similar level as full communication, although it should be unable to reach the same efficiency. Because heterogeneity is used to avoid plan overlap, the performance of stigmergy is expected to suffer as heterogeneity is reduced. A reduced frequency of updates should have the same effect. We expect it to be better than at least 90% of the baseline solutions at peak efficiency.

The lack of heterogeneity should have the same effect on the alternative without communication (RQ3), but to a larger degree. The solutions itself should not be able to reach the performance of the stigmergy method due to the lack of coordination, but is still expected to be superior to a majority of the baseline solutions. Without heterogeneity, the spacecraft should achieve roughly the same performance as a single spacecraft, as they all pursue very similar plans. With ideal heterogeneity, we expect them to beat at least 70% of the baseline solutions.

2 Background

2.1 The Space Environment

Our intuition tends to base itself on Earthbound assumptions which do not apply in outer space. We typically assume that things slow down and eventually stop due to gravity and friction, which does not happen to objects in space. We assume a fixed frame of reference, with a clearly defined “up” and “down”. We tend to assume that vehicles are stationary when their engines aren't running and no outside forces are acting on them.

In the environment of space, however, things are very different. Everything is always in motion. The location of a spacecraft changes by kilometres every second, so describing its current location is of little use. Instead, we talk about orbits.

Everything in outer space is in some kind of orbit. The Moon orbits Earth, which in turn orbits the Sun. The first thing to determine about an orbit is what it is orbiting, or what sphere of influence it is in. For most of this thesis, we will be describing objects in the Sun's sphere of influence. The only exception to this is when we talk about spacecraft orbiting an asteroid, in which case they are within the asteroid's sphere of influence.

The second thing to determine is the characteristics of the orbit itself. Typically, we use six characteristics defined by Johannes Kepler.[18] The size of the orbit is determined by its *semi-major axis* and its *eccentricity*, though it may be easier to understand in terms of *periapsis* and *apoapsis* (Figure 2). Periapsis and apoapsis are the closest and the farthest the object gets to what it is orbiting, respectively. The semi-major axis is simply the average of the periapsis and apoapsis. The eccentricity is a metric of how big the difference between the two is. An eccentricity near 0 indicates a circle-like orbit, while the apoapsis tends towards infinity as the eccentricity tends towards 1.

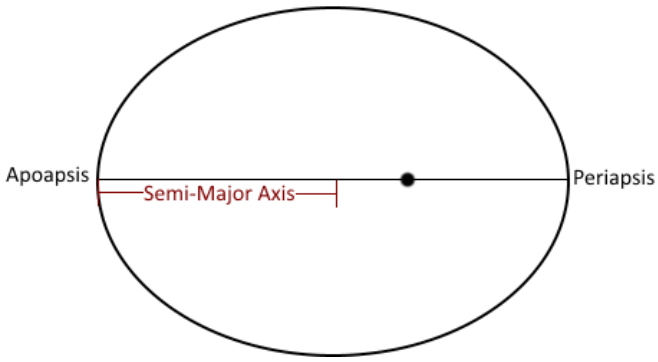

Figure 2: The basic orbital elements explained. The black dot is the orbit's centre.

Two more elements determine the plane of the orbit. For our purposes, this is generally the plane compared with the plane defined by Earth's orbit around the Sun. The two relevant elements are the *inclination* and the *longitude of the ascending node*. The inclination is simply the angle between the two planes. The *ascending node* is the point where the orbit crosses the reference plane heading “upwards”; usually taken to mean towards the Earth's northern hemisphere for solar orbits. The longitude of the ascending node is the angle in the Earth's orbital plane between the arbitrary chosen longitude of 0 degrees and ascending node. By convention, a line drawn from Earth through the Sun at Vernal Equinox is used as longitude 0 degrees.

The *argument of the periapsis* is the angle between the periapsis and the ascending node in the orbital plane. It determines where in the orbit the periapsis and, by extension, the apoapsis is. As

such it is the final factor in determine the exact size, shape and location of the orbit, as seen in Figure 3. So the only thing that remains is determining where in the orbit the object is at any given time.

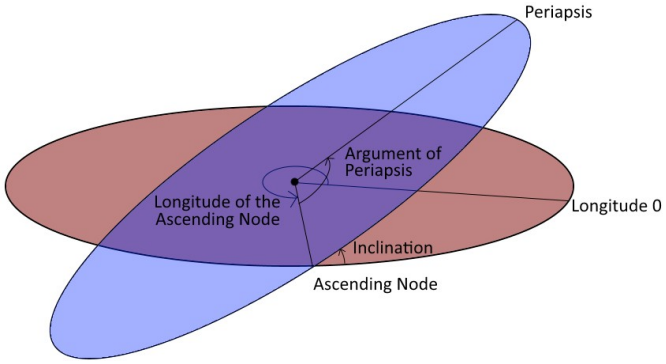

Figure 3: Three more orbital elements illustrated. The blue surface is the orbit being examined, while the purple is that of the Earth.

The *mean anomaly at epoch* describes the location of the object at a chosen time (the epoch). The mean anomaly is a mathematical construct which has the useful property of changing linearly with time. This allows the mean anomaly at any given time to be quickly determined. The mean anomaly can be readily used to determine the *true anomaly*, which is the angle between the object's current location and its periapsis.

The HIVE from the APIES mission, described below, demonstrates a number of special cases. First of all, its apoapsis, periapsis and semi-major axis are all at 2.6 AU, since its orbit is defined as circular. This means it has an eccentricity of zero, since the eccentricity measures the difference between apoapsis and periapsis. The inclination is also zero, as the HIVE is in the same plane as the Earth, which has no inclination by definition. With no inclination, there is no ascending node, so its longitude is typically assigned a value of zero (by convention). As the orbit is circular, the location of the periapsis is undefined, so its argument is similarly assigned a value of zero.

Due to the early stage of the APIES mission planing, its exact location within its orbit at a given time is not yet known. With a circular orbit, the mean anomaly of the spacecraft is identical to its true anomaly, as its is moving around the sun at a constant speed. The mean and true anomaly is usually defined by the argument of periapsis, which was just assigned a value of zero. The argument of periapsis is defined compared to the longitude of the ascending node, which is also undefined in this case. Thanks to the way these values are defined in relation to each other, the end result is that the anomaly is defined with respect to longitude 0 degrees (Vernal Equinox).

2.1.1 Transfer Windows

Space travel is very different from terrestrial navigation. With no friction or other major forces to affect the spacecraft, it will continue to coast along its fixed orbit unless its rocket engines are fired. Navigation is thus not simply pointing at your destination and turning the engines on. Instead, two engine burns are used. The first manipulates the spacecraft's orbit so that it will intercept the target asteroid (which is moving around the sun in a different orbit). Then, in order to not just keep flying past the asteroid, a second burn is used to match orbits with it. From then on, the two objects will orbit the sun together until another burn is used to change orbits yet again.

To successfully intercept an asteroid, the spacecraft and asteroid must arrive at the same place at the same time. As can be imagined, this is not trivial to arrange. The challenge of finding orbits which allow this is known as Lambert's Problem, after the man who first found a solution in the 18th

century. Specifically, Lambert's Problem is the problem of finding an orbit which goes from one given point at a given time to another given point at a later given time. The solution is mathematically complex, but can be near instantaneously calculated by a modern computer.

The engine burns expel propellant, exploiting Newton's laws to generate a force on the spacecraft.[19] The total change in velocity that a spacecraft can produce with its available propellant is called its ΔV (delta v) budget. Delta v is a shorthand for change in velocity. Each burn expends some amount of delta v to change the spacecraft's orbit.

While it is theoretically possible to transfer between asteroids at any time, the delta v costs are usually prohibitively high outside of rare opportunities. These opportunities are referred to as transfer windows.

Transfer windows are usually calculated by way of pork-chop plots.[20] These two dimensional graphs plot the transfer cost as a function of departure time and transfer time. For manned missions, the goal is usually to minimise transfer time, while keeping departure time and transfer cost within acceptable limits. For unmanned missions, and especially multi-target missions like ours, the goal is to minimise transfer cost, while keeping total mission time within acceptable limits.

The pork-chop plot is used to find the minima at the peaks of the contour map. Typically, the x-axis represents time of departure, while the y-axis represents either arrival time or time of flight. In the example shown in Figure 4, the Mars Reconnaissance Orbiter could have left in early August and arrived in late February 2006 with the bottom launch window. If the launch was delayed to early September, it could take the top launch window and arrive in October instead. In this example, the red lines indicate fixed time of flight with one line for every 25 days difference in time of flight.

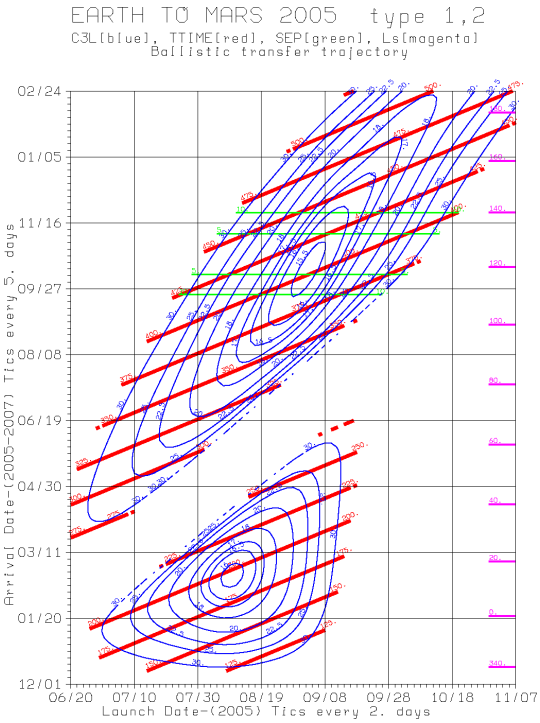

Figure 4: Shows an example of a simple pork-chop plot. There are two maxima, one in the middle of each "pork-chop".[45]

Unfortunately, while Lambert's problem is easy for a computer to calculate for a given set of departure and arrival time, a two dimension grid search is needed to actually locate transfer windows. Depending on the desired resolution, this can quickly get computationally demanding. In Figure 4, NASA has sampled departure dates 2 days apart, and arrival dates 5 days apart.

2.2 APIES

The APIES mission[4] features one Hub and Interplanetary VEhicle (HIVE) and 19 BElt Explorers (BEEs). The mission focuses on known technologies and very little redundancy to reduce weight. It will be launched on a Soyuz-FG/Fregat rocket, a reliable 15 year old Russian design. From there, the HIVE makes its way towards the belt via a Mars gravity assist and conventional engines, functioning as a mother ship for the BEEs.

The BEEs are designed with an extreme focus on component reuse. For example, the main propulsion system and the steering thrusters use the same propellant tank to save weight. Similarly, the radio antenna doubles as a radar during the flybys and the same lens is used for both the camera

and the IR spectrometer on board. There is no redundancy on the individual BEEs, because losing a couple does not jeopardise the mission. As a result, each BEE weighs only about 45 kg, while carrying sufficient instruments to detect the shape, mass, and composition of an asteroid. It barely has enough power to run one system at a time, however, so it has to choose which system to have active at any one time: engines, sensors, or communications.

The HIVE carries all the propellant needed to reach the asteroid belt in the first place, so it weighs in at 540 kg. Heavy solar panels and long range communication systems make up most of the remaining mass. The HIVE coordinates the swarm and facilitates messaging between Earth and the BEEs, as well as internal communication among them.

The HIVE brings the BEEs out to a circular orbit 2.6 AU away from the sun, set in the same plane as the Earth's. This orbit was carefully picked by the APIES mission designers to balance a number of requirements. Propellant limitations had to be balanced with the local density of the asteroid belt and access to variety of asteroid types. At 2.6 AU, the HIVE is placed right in the middle of the main asteroid belt, close to where the density is highest. It is also in the transitional region between the inner asteroid belt, dominated by bright S-type asteroids and the outer asteroid belt, which is dominated by much darker C-type asteroids (see section 2.3). This ensures a wide variety of asteroids to sample at the target location.

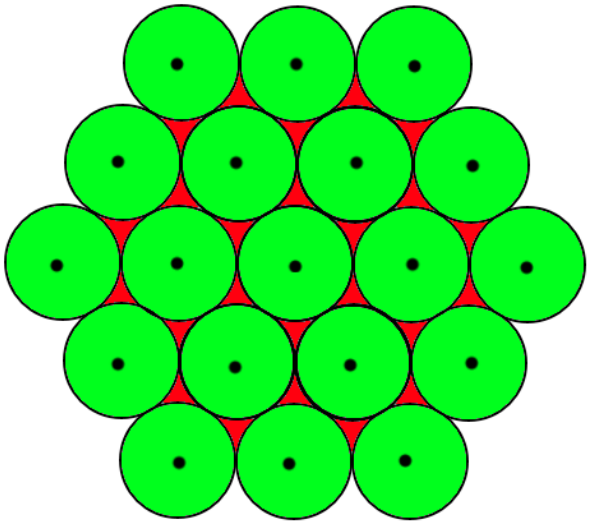

Figure 5: Shows how the BEEs of the APIES mission take up formation. The green areas are unambiguously patrolled by a single BEE. Asteroids which pass through the net in the red areas have three possible interceptors.

Once in the asteroid belt, the BEEs are launched from the HIVE and take up formation around it (Figure 5). The BEEs are deployed individually whenever a target of interest presents itself. Each BEE is assigned an interception zone 0.05 AU across, and makes a flyby whenever an asteroid is about to pass through that area. The spacecraft captures visual, IR, and radar data during the flyby, allowing for estimates of the asteroid's volume and mass to be made. Within 16 minutes, the flyby is finished and the asteroid is disappearing into the distance. The BEEs stay relatively far from the asteroid, with a closest approach around 20-25 km away from the surface. The mission hopes to investigate a hundred asteroids during six years.

2.3 The Asteroid Belt

When we imagine an asteroid belt we often think of the dense fields of floating rock depicted in mainstream science fiction.[21] But this this not how the Asteroid Belt actually behaves.

Not that there is any scarcity of asteroids. At the time of writing, well over half a million asteroids have been identified and tracked.[22] Estimates of how many there are in total vary significantly, but its generally assumed to be in the order of tens of millions. [23]

But the asteroid belt covers a massive area, as seen in Figure 6. The inner and outer edges are around 200 million kilometres apart, and the belt covers a roughly doughnut shaped area with a 2 500 million kilometre circumference. There is a spectacular amount of space available. Even with millions of asteroids, there are trillions of cubic kilometres of empty space for each of them. [23]

While asteroids are, on average, millions of kilometres apart, they are also all moving around the sun at 15-25 kilometres per second. Each has its own individual orbit, taking 3-6 years to go around the sun. Close encounters with Mars, Jupiter, and each other can alter the orbits of asteroids significantly and collisions are common on a geological time-scale. Luckily for us, the asteroid belt appears a lot less dynamic over the course of a decade long mission. [23]

While collisions do occur fairly often, most are relatively serene events. Examples include asteroids in near identical orbits bumping into each other at relative velocities of only a few meters per second and pebbles impacting with a kilometre sized object. Such events would not pose a threat to an orbiting spacecraft. The cataclysmic collisions seen in popular culture are unusual enough that it is doubtful if even a single such event will occur anywhere in the belt during the years of the mission. With a very low chance that asteroid collisions will affect the mission at all, we can safely exclude it from our model.

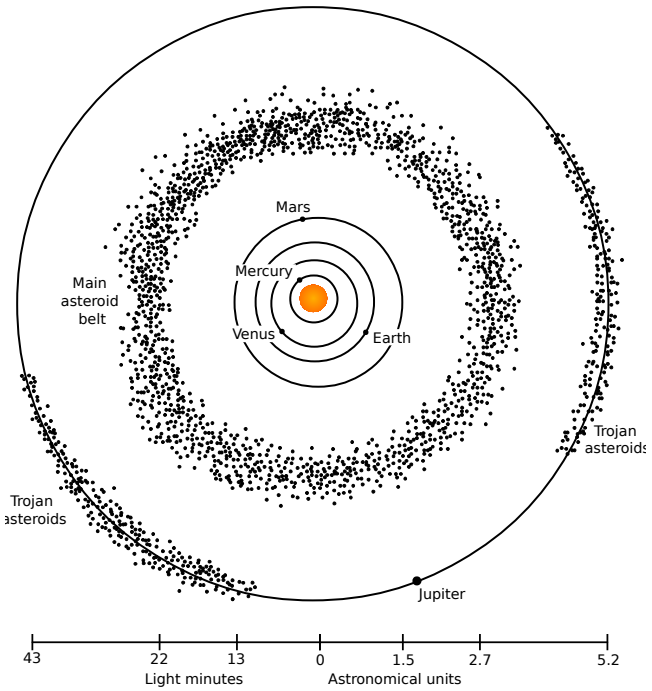

Figure 6: A diagram showing the location and size of the main groups of asteroids in the solar system.[46]

Encounters with planets and large asteroids can change an orbit enough to befuddle attempts to map the asteroid belt. This does not affect the mission enough to warrant special modelling, however. Fortunately, encounters only take place when the asteroid and planet are at the same place at the same time, which as described rarely happens. Additionally, any such asteroids can be removed from the model before planning as such encounters can be predicted decades in advance.

2.3.1 Asteroid Naming

The majority of asteroids currently have temporary names chosen by the automatic system which first discovered them. These typically consist of the year, a one letter code indicating when in the year the observation was made, and a letter for how many asteroids have thus far been detected in that period. They go through the letters alphabetically, though skipping I to avoid confusing it with 1. In recent years, more than 25 asteroids are usually discovered in any 2 week period, so a number is added to the end to indicate how many times they have cycled through the alphabet in that period. When a later set of observations confirm that the calculation of the object's orbit was accurate, it is given the next available number as a prefix to its temporary name.[48]

As an example, take 210280 2007 TS39. It was discovered in the beginning of October 2007, as indicated by the “T” and the “2007” in its name. The 39 indicates that this is the 40^{th} TS asteroid of that year (the first being TS, the second TS1 and so on). S is the 18^{th} letter of the alphabet when I is removed and the I-less alphabet is 25 letters long. That makes TS39 the $18+39\cdot25 = 993^{rd}$ asteroid discovered in early October, 2007. It is also the $210{,}280^{th}$ asteroid to ever get its orbit confirmed.

The temporary designation “2007 TS39” may eventually be replaced, as the International Astronomical Union is working to find permanent names for every asteroid. However, the discoverer of the asteroid has a monopoly on suggesting names until 2017; ten years after its initial

discovery. After that, the public may suggest names as well.

2.3.2 Asteroid Classification

Asteroids are not easy to classify. Assigning a classification to an asteroid requires detailed observations of the asteroid over a wide frequency spectrum (largely infra-red, visual and x-ray). As such, most known asteroids remain unclassified.

Depending on the classification system used, there are anywhere between 3 and 12 main classifications of asteroids, each with multiple sub classes. The earliest classification systems were based on how brightly the asteroid shines in the night sky (corrected for asteroid size and distance from Earth). More recent attempts divide asteroids by composition; a task made difficult by the lack of hard data. There are, however, three classes of asteroid which have been present in nearly every classification system which are worth taking a closer look at. These classes are believed to make about 95% percent of all known asteroids and virtually all asteroids in the APIES mission area, as seen in Table 1.

Asteroid Class Distribution

Asteroid Class	*Proportion among Asteroids at 2.6 AU*
C-type	~18%
S-type	~63%
M- or X-type	~19%

Table 1: Asteroid classes present in the APIES mission area.[47]

The most common class of asteroid is the Carbonaceous or C-type asteroids. As the name suggests, these are believed to contain significant amounts of carbon. They are challenging to detect due to their coal black outer surface, but are believed to make up about 75 percent of all asteroids. It is believed that carbonaceous chondrite meteorite samples recovered from impact sites are pieces of C-type asteroids. This would indicate that they are a frozen mix of organic chemicals and water ice condensed out of the cooler regions of the primordial solar nebula. But, as no samples have been returned from any C-type asteroid, we will not know for sure until Hayabusa 2[15] returns to Earth in December 2020. C-type asteroids dominate the outer parts of the main belt, from about 2.7 AU out from the sun, but are found all over the solar system. In the area explored by the APIES mission, the proportion of C-type asteroids is believed to be about 20%, with the number growing as one heads further out.

The second most common class of asteroid is the Stony or S-type asteroids. These have seen more study than C-type asteroids. They shine brightly and are therefore very easy to detect and track. The only successful asteroid sample return to date, the first Hayabusa mission[14], also targeted one. S-type asteroids seem to be made out of stone, mostly pyroxene and olivine. They also contain chondrules, round grains of other materials formed from molten droplets. S-types are most common in the inner asteroid belt, constituting almost 80% of the asteroids around 2.1 AU out, but can be found just about everywhere in the inner solar system. At the APIES mission area, the proportion of S-type asteroids is on the way down, being just over 60%.

The Metallic or M-type asteroids were originally thought to be made of metals. While some of them almost certainly are, others have been found to have far too low densities to be made of metallic compounds. The actually metallic asteroids are largely composed of nickel-iron alloys, with varying amounts of stones and other metals. The composition of the non-metallic M-type asteroids remains unknown. M-type asteroids do not seem to cluster into any particular region, though are noticeably more common in proportion to other types in the main asteroid belt than elsewhere in the solar system. Due to the predominance of non-metallic M-type asteroids, some classification schemes rename the class as X-type asteroids to avoid implying a metallic nature.

While these three classes make up the vast majority of the asteroids in the solar system, the remainder have proved difficult to classify. Not every classification scheme uses all these classes

and some asteroids fall into different classes depending on which scheme is used. D-type asteroids fall somewhere between the C and S class; they are carbon rich, like C-types, but contain silicates and very little water, like S-types. V-type asteroids are believed to be pieces of the large asteroid 4 Vesta, dislodged by impacts from other asteroids over the years. The O-type asteroids are believed to be the cause of a relatively common meteorite type, though 3628 Božněmcová is the only discovered O-type to date. The P-type asteroids are typically found in the outer edges of solar system, and appear to have an icy core covered in rock and organic chemicals. T-types, L-types, and K-types all have a clear and distinctive absorption spectra, but their composition remains completely unknown.

3 Related Work

Surprisingly, little research can be found investigating this type of problem and none in terms of deep space missions. Research mostly focuses on single spacecraft missions and satellites in low Earth orbit. Inspiration must therefore come from other fields.

3.1 Optimal Foraging

The problem appears similar to optimal foraging at first glance, which concerns itself with the foraging behaviour of animals in nature. Much can be said about optimal foraging and its details,[24,25] but it can be summed up in a few points. The optimal strategy is found by maximising the animal's rate of net energy intake given its biological limitations. This is done in pursuit of gaining a large enough surplus for activities like breeding. An animal searches for nourishment in an area and upon finding a potential food source, chooses whether or not to pursue it. The choice depends on various factors, most importantly the handling cost versus the energy reward.

In our case, the spacecraft have a non-replenishable reserve of propellant. Energy is simply harvested from the sun, with a rate of intake mainly determined by the relative angle of the solar panels. Unlike optimal foragers, the BEEs are unable to search for asteroids. The problem is to reach them efficiently, not discover them. Rather than maximising the rate of intake, the goal is to maximise the lifetime scientific return. Hence, we cannot use optimal foraging techniques.

3.2 Bidding

Traditional multi-agent systems tend to use metaphors related to economics or political theory. Agents bid or vote on tasks, based on their internal models of the world. An agent might see that it has a low cost transfer window to a high value asteroid and bid a relatively large amount on being assigned to visit that particular target. The agent with the highest bid is assigned to visit that particular asteroid, while the remaining agents look for other opportunities. [26]

The main problem with such systems is the large amount of required communication. Even if there is only one bidding round, each agent needs to inform the others of its bid for every asteroid and receive the others' bids in return. This can be done through an intermediary (the HIVE comes to mind) or directly, though that would require even more communication overhead.

Unless sophisticated algorithms are used for choosing what to bid, such agents tend to be short sighted and greedy. While bidding makes a decent decision making algorithm, it is not a planning algorithm in and of itself. Ideally, we would want an algorithm which handles both planning and coordination.

3.3 Route Planning

What about algorithms for finding the shortest route? There are several reasons why these algorithms cannot be used directly. First of all, the goal is not to find the best route between two given nodes. Rather, we want to find the best route from a specific node to any other node. As the number of nodes is at least half a million, this quickly makes the problem intractably large. Second, the nodes change value depending on whether other nodes have been visited. The algorithms would have had to be modified to account for this. Third, each of the multiple spacecraft would need a distinct route with minimum overlap. This means that any such algorithm must at least be adapted to our problem, if this can be done at all.

The vehicle routing problem[27] is another problem which looks similar to our problem. In essence, it answers the question of how a fleet of vehicles should be routed to service customers at different

locations at minimal cost. Many variants[28] exist, but the *fixed fleet open vehicle routing problem with capacity limits and time windows* seems to be the closest.

There are, however, a few complications. The first is that each customer (asteroid class) has a large number of service locations (asteroids), and visiting one location reduces the need to visit the remaining ones. The vast distances and number of potential locations to visit means most can never be reached. The cost of communication in time and energy is also not considered in the standard versions of the problem. Another issue is that the vehicle routing problem is a general version of the travelling salesman problem. As the travelling salesman is the text book example of an NP-hard problem, that makes the vehicle routing problem NP-hard as well.

3.4 Genetic Algorithms

Genetic algorithms[29,30] can be used to find approximate solutions to NP-hard problems if a good genetic representation of the solution space can be chosen. That is, if a route through the asteroid belt can be represented as a genotype and manipulated by genetic operators. We also need a mechanism to translate a genotype to a phenotype, in our case to produce a node-to-node route. Finally, we need a metric to evaluate each phenotype and give it a score representing its fitness. The genetic operators should be chosen such that they maintain and evolve a population of ever more fit genomes; hopefully while maintaining sufficient genetic variation to cope with any changes in the fitness metric.

Fitness metrics allow the relative fitness of two genomes to be compared. A fitness metric assigns a fitness value that represents some absolute fitness to each genome in the population. When selecting genomes for breeding, these values are used as a basis for the selection process. Naturally, desired behaviour should increase the fitness value while unwanted behaviour should reduce the fitness value.

There are several reasons to use a genetic algorithm for this problem. The first is that genetic algorithms can approximate solutions to NP-hard problems, producing a reasonably good solution even when it is infeasible to find the optimal one. Additionally, a population of solutions is kept and iteratively improved upon. If the circumstances change, the population of solutions should be able to quickly adapt. In other words, if a spacecraft receives information that an asteroid it plans to survey has already been visited, solutions involving that asteroid will receive a lower fitness. Alternative members of the population will receive a relative boost in fitness. Mutations and crossover may also eliminate the asteroid from the population completely within a few generations.

The most important reason for why a genetic algorithm works well is that it does not require specialised domain knowledge. This allows us to spend more time focusing on the impact of communication. We will use a fairly straightforward genetic algorithm to evolve reasonably good solutions. Our priority is on developing a more complex fitness function that will handle the varying amount of available information. However, algorithms tailored to the problem would probably provide better solutions in the long run.

3.4.1 Selection Methods

A variety of selection methods have been proposed over the years.[30,31] The simplest is probably *Elitism*, which simply selects the best *n* options available. Typically, it is used when *n* is large; making the selection more a matter of pruning out bad solutions from the population than about selecting good ones.

Tournament selection picks out a small number of random options and returns the best among them; repeating this process *n* times. While this is a bit more computationally complex than Elitism, it is a lot faster, as it does not require that the entire population be sorted in order of decreasing fitness.

Roulette-wheel selection chooses genomes to keep with a probability based on its fitness score. It is also known as fitness proportionate selection. It can be imagined as a roulette wheel being spun, where each slice is sized based on the fitness of the individual.

A variant of the roulette-wheel selection is *stochastic universal sampling*, which uses multiple evenly spaced markers instead of the single one traditionally used in roulette. This ensures that some of the poor solutions survive, which may be desirable. It provides more genetic diversity while still selecting more good solutions than bad.

3.5 Stigmergy

Tripp and Palmer[7,32] have investigated the use of stigmergy for a swarm of satellites orbiting the Earth and photographing the surface. The swarm receives new tasks with a value and deadline, which are incorporated into the swarm's overall plan. Stigmergy is used to create a form of aggregated common knowledge of the swarm's activities. Each satellite reports its current plan, which is incorporated into the aggregated statistics of the virtual environment. At regular intervals, they all receive aggregated information in return. From this, they can see which tasks have been completed, as well as which are scheduled to be completed in the near future. The method produces overall satisfying results with high efficiency, low overlap, high degree of adaptability, low need for communication. The largest advantage, however, is the absence of satellite to satellite communication.

In order to prevent overlap in the tasks chosen by the satellite swarm, Tripp and Palmer investigated the use of heterogeneity. In their method, each member of the swarm is given a unique combination of four personality characteristics. These four characteristics alter how the satellite assigns fitness values to their future plans, and hence which tasks it will perform. The characteristics used by Tripp and Palmer are:

- *Greedy*: makes the agent prefer high value targets. Since high value tasks presumably have been given a high value because it is important that they are done, having a behaviour which prioritises the important assignments is a logical first choice. Its also assumed that if there is unavoidable overlap, it is better that said overlap is in regards to important tasks rather than low value ones.
- *Considerate*: makes the agent avoid areas of previous overlap. As the satellite swarm is in various orbits around Earth, agents can assume that areas of previous overlap are regions which multiple spacecraft pass over on a regular basis. Hence, for Tripp and Palmer's scenario it makes sense to have a behaviour which avoids such areas in order to favour regions the satellite is uniquely equipped to service.
- *Proactive*: makes the agent avoid future areas of overlap. Having information about the plans of other agents is only useful if it actually influences the agent's decision. The proactive behaviour causes agents to reconsider plans that involve tasks other agents intend to do.
- *Obstinate*: makes the agent avoid changing its plan. The obstinate behaviour encourages predictability. If the agent has told the other agents in the swarm that it intends to do a certain task, it can be assumed that the other agents will to some extent modify their plans to take this into account. If the agent then instead chooses a new and different task to do instead, the old task may go uncompleted.

By weighting these behaviours differently in the fitness function, agents end up with different priorities. Some are erratic and adaptable, filling in for whatever tasks are left over once the others have settled on their plans. Others are predictable, picking high value targets and ensuring that the

most important tasks get done. While homogeneous agents would still be able to coordinate, the benefit of heterogeneity has been well demonstrated over a number of papers.[7,32]

3.6 Monte Carlo Simulation

Monte Carlo Simulations[33] are a popular way of mapping complex solution spaces. The idea is that picking many random inputs and examining their aggregated outputs provides a good idea of what outputs are likely and not. In our case, by having the BEEs pick random transfer windows, we can examine the distribution of total scientific return to get an idea of what possible solutions exist. And since each random simulation is done very quickly, we can run a lot of them, providing a good mapping of the utilities in the solution space.

The main weakness of Monte Carlo Simulations is that they may have difficulty locating highly unlikely outputs. If the graph of available transfer windows has few edges or a high degree of clustering around a small group of asteroids (many roads leading to Rome, so to speak), it may be rare to find a solution in which all 19 BEEs pick a disjointed set of asteroids to explore. We don't consider it particularly likely that we will end up in this situation, as it is expected that there will be hundreds of available transfer windows from any given asteroid.

What we get out of a Monte Carlo Simulation is a probability distribution of a random solution having a given utility. This provides a good baseline for later simulations, as we can examine how our solution compares to just choosing randomly. We can also see how many percent of the possible solutions are better than the solutions generated by our evolutionary algorithm. These properties allow us to use Monte Carlo as a control group, providing results which do not rely on an evolutionary algorithm.

3.7 Nelder-Mead

The Nelder-Mead technique[34] is a descent algorithm designed to locate minima in multi-dimensional problems with as few evaluations as possible. It only evaluates the target function at a few points per iteration, and typically converges relatively quickly. The algorithm works by manipulating a special shape called a simplex; moving and reshaping it towards the minimum.

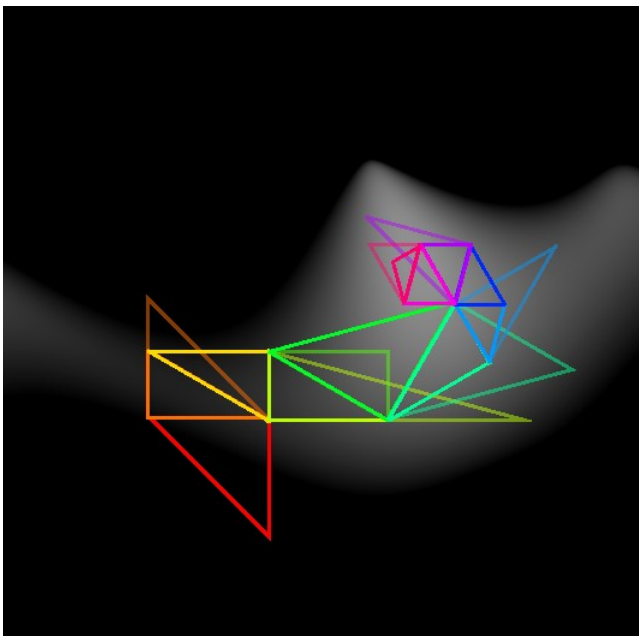

Figure 7: A hue based illustration of the Nelder-Mead simplex moving towards a minimum.

A simplex is a convex shape with as few as possible vertices while still covering a range of values in every dimension of the search space. An *n* dimensional search would have a simplex with *n+1* vertices. This is because a one dimensional search space would need two vertices to cover a range, and each additional dimension would need one more vertex to cover it. For a two dimensional search space, this simply means a triangle. In generally terms, the Nelder-Mead algorithm seeks to move the worst vertex in the simplex to a better location in each iteration of the algorithm.

Nelder-Mead is based on an older algorithm which used a combination of reflecting a point across

the average of the other points in the simplex and moving all the points in the simplex closer to the best one to work its way across the solution space. Nelder-Mead adds a couple of optimisations in order to speed up convergence. If the reflected point is better than any other point in the simplex, then it assumes it is on the right track, and expands in that direction, testing a point further out. If, on the other hand, the reflected point is worse than every other point, it contracts the worst point in towards the average of the other points. It only goes to the computationally expensive process of shrinking the whole simplex towards the best point if this too fails to find a better point. The process is depicted in Figure 7.

The main benefit of this method, compared to other descent algorithms, is that it does not require the derivative of the value function to work. For complex value functions, like the delta v cost of transfers, this is a major advantage. The derivative is not necessary known, and it may be far more complex to compute it than to calculate the function's value at a few additional points. This also makes Nelder-Mead particularly desirable for high dimensional problems.

4 Methodology

By the APIES plan, described in section 2.2, the BEEs will take up formation around the HIVE and form a drag net of sorts 0.25 AU wide. BEEs are dispatched to intercept any asteroid which happens to pass through the net, with the goal of getting within about 20km of it as it passes. There are advantages and disadvantages to this technique.

The main advantage is that it uses as little propellant as possible per interception. On the other hand, it offers little choice about which asteroids to intercept. They simply have to take whatever opportunities come along. Which means that if, by pure chance, a limited number of a certain type of asteroid crosses the net, the scientific utility of the mission suffers.

An alternative would be to actively seek out asteroids. This requires more propellant per interception, which means less asteroids visited over all. On the other hand, not only can more interesting asteroids be selected, but each can be studied in more detail. By going into orbit around an asteroid, rather than just a swift flyby, a more detailed examination of its mass and composition can be made with the same instruments.

After careful consideration, we have decided to analyse this alternative architecture instead of the one proposed by APIES. In addition to the reasons already discussed, the value of communication is easier to determine in the more chaotic asteroid selection scheme than in a simple net. In APIES, each spacecraft has its own intercept zone, so there is rarely any ambiguity about which BEE will perform the flyby. In addition, the orbiting method is more widely used in other missions, including PAM/ANTS[3] and Dawn[17].

We have further decided not to investigate the robustness of the APIES plan, which would rely on intelligent replanning and the redundancy of having 19 BEEs to overcome the relative frailty of each individual BEE. Hence, we do not require our model to include sudden failures or other eventualities. While there are some interesting questions in regards to how well the APIES plan can handle failures, these are tangential at best to our primary goal of determining the value of communication.

4.1 The Experiments

Specifically, we plan to do the following experiments based on our research questions (from section 1.3):

Exp 1 Our first and most basic experiment is a Monte Carlo simulation of the mission. This will involve simulating thousands of randomly generated routes to provide a baseline against which our AI can be compared. It naturally does not use the evolutionary planner module as such, though it can generate its routes through the same code used by the evolution to generate random genomes.

Exp 2 The second experiment assumes that the BEEs have unlimited possibilities to communicate, as represented by the co-evolution of plans. Every genome in every generation is paired up with those from other BEEs to see how well they can work together. We refer to this experiment as the Full Communication experiment or scenario, and it corresponds with RQ1.

Exp 3 The third experiment limits communication to occasional stigmergy updates (section 3.5) in which every BEE is provided aggregate information about the actions and plans of the rest of the swarm, updating their internal plan accordingly. This requires pausing the simulation so that the BEEs can reconsider their routes whenever more information becomes available. This is the Stigmergy experiment or scenario, based on RQ2.

Exp 4 The final experiment, based on RQ3, has no communication at all, relying instead on the use of heterogeneity in the fitness functions to avoid overlapping routes. In this No Communication scenario, each BEE plans its route separately, without any information about the other members of the swarm.

4.1.1 Scientific Return

Having determined what to study, the next step is to ascertain how to measure success. The goal of the APIES mission is to learn as much as possible about the asteroid belt given the resources available. The best metric by which to measure our success is thus the amount of useful data collected. Not all data is equally useful, however. As the amount of data available about a class of asteroid increases, one reaches a point of diminishing returns. Once the density and surface composition of a class is known with a fair amount of certainty there is not much more that can be discovered with the limited instrumentation available.

Ideally, the BEEs should examine a multitude of different asteroid classes. The first close examination of a given class provides a wealth of new data. The second provides a valuable point of comparison to help determine exactly which features are characteristic of the class and which are individual to the asteroid. The third and consecutive asteroids provide less and less new information about the class, although later asteroids may still reveal surprises. Revisiting an asteroid provides no new information, as our version of the APIES mission spends enough time around each asteroid to fully study it with the available instrumentation.

Each specific asteroid has its own intrinsic value, based on how interesting that asteroid is for the scientists of the 2020s. Since we do not know what asteroids they will find interesting, we have chosen to provide each asteroid with a random intrinsic value instead. This provides us with the desired differentiated environment without requiring the manual assignment of half a million values.

As such, we will regard the utility of the asteroid to be two-fold. First, there is a classification utility, which depends on how many asteroids of that class has thus far been explored. Second, there is an intrinsic value, which is unique to each individual asteroid. Revisits provide no added utility of either type.

The BEEs are expected to spend a lot of time transmitting their findings to the HIVE. However, we are saved from having to model dedicated transmission times by the fact that they are also expected to spend a lot of time orbiting an asteroid while waiting for a transfer window or coasting between asteroids. As per the unmodified mission described in section 2.2, this “dead” time is used for information transfer instead of the craft simply going dormant. However, communication is not free and there is a propellant cost associated with keeping their antenna aligned with the HIVE. As such, a small amount of propellant is subtracted for each asteroid visited.

4.2 Transfer Windows

For a given departure and transfer time, it is computationally simple to calculate transfer cost. However, this is a transfer between two given asteroids (origin and destination), so the number of calculations scale quadratically with the number of asteroids examined. Up to half a million asteroids may be within reach, and we have six years worth of time to search along both axis. As a result, online generation of transfer windows is not feasible with our available computing power.

Instead, a set of locally optimal transfers will be produced offline and used throughout our experiments. A slope climbing algorithm is used to find the peaks of a pork-chop plot without having to calculate the cost of every possible transfer. Furthermore, an asteroid is only considered as a viable origin asteroid if a path has been found by which it can be reached by a BEE. This

should cut the computational time down to within feasible limits.

It is possible that, even with these optimisations, the time required to generate a complete map of viable transfer windows is too large to calculate on a reasonable time-scale. In which case we may be forced to use a subset of the known asteroids within the mission area, and increase the BEEs' available resources to compensate for the reduced density of asteroids.

When the calculations are complete, what remains is a set of viable transfer opportunities, each with an associated cost. As each window connects a pair of asteroids, this can be modelled as a graph structure with the asteroids as nodes and the transfer windows as edges. However, unlike typical graphs, the edges can only be traversed at a specific point in time. At any rate, the graph is far simpler than a physical model of the asteroid belt would have been.

4.3 Model

Our model is a simplified representation of the asteroid belt described above. The goal is to maximise a value V_{sum} representing the scientific return of the mission. The first time an asteroid is visited, the asteroid's randomly assigned scientific value V_a is awarded for surveying the asteroid itself. Additionally, there is a reward V_c for surveying examples of the asteroid's class. The first two times members of the asteroid's class are visited, the award is the initial value of the class C_v. Subsequent visits award $V_c(n)=C_v \div n$, where n is the number of unique members of the class previously surveyed. The total value received from an asteroid is thus V_a+V_c. The scientific return is highest when the BEEs visit as many unique high value asteroids as possible while ensuring that several different classes are visited, preferably high value ones.

The asteroids themselves are contained within a set of nodes $a \in A$. Associated with each asteroid in A is a set of time dependent edges, W_a, describing all transfer windows originating from that particular asteroid. $W_{o,d}$ describes the set of transfer windows from an asteroid a_o directly to another asteroid a_d, $a_o \neq a_d$.

A transfer window $w \in W$ consists of: origin and destination asteroids a_o and a_d; time of departure and arrival $t_o < t_d$; and propellant cost $p > 0$. From this, the time of flight $T_{travel}(t_o, t_d) = t_d - t_o$ can be deduced. The propellant cost is the calculated cost of the transfer, plus a small additional constant P_{comms} representing the communication costs. These parameters are summarised in Table 2.

Transfer Window Parameters

Name	*Symbol*
Origin Asteroid	a_o
Destination Asteroid	a_d
Time of Departure	t_o
Time of Arrival	t_d
Propellant Cost	P

Table 2: Parameters used by the Transfer Windows.

A spacecraft $s \in S$ represents our BEEs. Each spacecraft has some amount of remaining propellant s_p and a *location*. s_p is reduced every time the spacecraft makes a transit, and starts at a universal P_{max}. At any given time, a spacecraft's *location* is either in orbit around an asteroid a or using a transfer window w.

Spacecraft Parameters

Name	*Symbol*
Propellant Remaining	s_p
Current Location	*location*

Table 3: Parameters used by Spacecraft.

The spacecraft takes a route *r* through the asteroid belt, which is described as a list of transfer windows $r = \{w_1, w_2, w_4, \ldots, w_n\}$ with the following restrictions:

- For any w_i and w_{i+1}, w_i must arrive at the origin asteroid of w_{i+1} before the departure time of w_{i+1}.

- The sum of propellant costs $p_{\text{sum}} = \sum_{i=0}^{n} p_i$ is always limited to the maximum capacity of any spacecraft $p_{\text{sum}} \leq P_{max}$. If a new route is considered after mission start, the restriction becomes $p_{\text{sum}} < s_p \leq P_{max}$, as the spacecraft has presumably already expended some propellant.

4.4 Simulation

We chose to use an event based simulation, in order to minimise its computation complexity. Our simulator consists an ordered list of events which are resolved one by one until the mission is complete. Some examples of events include: departures from asteroids, arrivals at asteroids, stigmergy updates, and the end of the mission. Each event has a given time in which it occurs. Events can spawn new events (departures spawn arrivals or stigmergy updates spawning complete new plans) and may or may not have an associated spacecraft.

This very simple simulation does everything we need it to, and is very light-weight. It is also fairly simple to implement. Spending less time on a complicated simulation gives us more time to implement the evolutionary algorithms, and a faster simulation lets us do more trials in the limited time available.

4.5 Evolutionary Planning

As explained in section 4.1, there are three main communication systems in our study: Full Communication, Stigmergy, and No Communication. All three have the same overarching goal and solution: to maximise the scientific return of the mission by ensuring the spacecraft visit high value targets and do not have overlapping paths. However, the tools available to the spacecraft differ vastly in the three cases.

We would like to reuse as much of our code as possible between the different trials, both to avoid skewing results in favour of whichever version of the code works best and to reduce work load. Ideally, the only difference in code for the different trials should be the fitness function, and how much information it has access to when assigning the fitness of a plan.

The Evolutionary Planning module makes up the bulk of our simulation code. As described above, our simulated asteroid belt is very simple and light-weight, once it has been generated. The evolutionary planning is what will likely take up the bulk of the run time.

When the BEEs have full communication or no communication at all, the planning is all done before the mission, and the simulation, starts. As we will see in section 4.5.2 below, however, the co-evolutionary nature of the Full Communication experiment means that the fitness evaluation will involve running the simulation multiple times. Stigmergy is slightly different, as the simulation will have to pause occasionally to allow the BEEs to update their plans with the new information available.

4.5.1 Genome

The most important part of a successful evolutionary algorithm is picking a good genotype. No matter how clever the evolution is, it only works if the genotype maps well to the solution space. The two main categories are direct representation, where there is a one-to-one correspondence between genotypes and phenotypes, and indirect representation, where the genotype provides the parameters used to generate the phenotype.

Indirect representation can be easier to implement as it allows well established genetic operators to be used, even on unusual problems. However, with a direct representation we can more easily be

sure that the entirety of the solution space is covered, allowing better solutions to be located. On the other hand, it typically requires more complex and customized genetic operators, which may slow down the evolutionary process compared to the well established ones.

A naive indirect representation would be to have each gene be the number of transfers to ignore before taking the next available transfer opportunity. In order to handle overflow, this would have to be modulo the number of possible transfers from the asteroid. This is, naturally, a terrible genome choice. Any amount of modification to the genome, be it from crossover or mutation, would completely scramble the remainder of the route in unpredictable ways. The result is that the child of two excellent routes would likely be worse than either parent, preventing fitness from rising over time.

While a more complex indirect representation could be devised to negate such issues, the amount of effort which would have to go into crafting and verifying it would simply not be worth it in the short time we have available. Especially as we were quickly able to locate a suitable direct representation. By directly using the spacecraft's route as the genome, we are able to minimise the translation process from genotype to phenotype. The process simply involves cutting the genome at the point at which the spacecraft runs out of propellant.

In addition to requiring very little computing power, this makes our genomes human readable enough to simplify debugging. As this takes the form of a list of transfer windows (with a total list length in the dozens at most), it is memory efficient. As described below, this choice has led to increased complexity of the selection process. The risk involved has been managed by picking a method which rates the fitness of every possible child, which means we will not lose any good solutions; even if the process is a bit slower as a result.

4.5.2 Fitness Metrics

With the genome chosen, the next most important part of an evolutionary algorithm is the fitness metric. As described in section 4.1, we have a number of different fitness metrics depending on how much information each BEE has access to. The three levels are: Full Communication, Stigmergy, and No Communication.

Full Communication

Full Communication allows the spacecraft to create one collective plan to avoid unnecessary loss of performance caused by overlapping routes or visiting low value targets. A simple and straightforward fitness metric to do this is simply to simulate the same scenario many times, but with different sets of genomes. In a simulation, each spacecraft follows the plan of one genome, which is awarded a score equal to the total scientific return of the scenario. The simulations are run such that every genome is measured several times, and its final fitness is the average of the scores it has been awarded.

When genomes are drawn for evaluation, one is drawn from each BEE. This allows each population to specialise and conquer a niche. This also reduces the complexity of testing by limiting the potential number of genome combinations to test.

A problem is that a set of genomes may produce a superior scientific return when paired together, but not when combined with other genomes. Although the particular selection would be a superior overall plan, the fitness value would still be close to average. There is no guarantee that the particular selection will ever be made again, losing the superior plan. On the other hand, genomes that do not work well with others may not be of any interest regardless of the one time scientific return, as they may not be robust to change.

The problem can be solved by combining each selection into a super-genome. Unfortunately, time

constraints make implementing two different genomes, genetic operators and selection processes too time consuming. The multi-population method allows us to reuse code between the three levels of communication, while the super-genome method only works with Full Communication.

The precise description of the chosen method is co-evolution of n populations, one for each spacecraft. To evaluate the current generation of populations, one genome is drawn from each population at random. The n randomly drawn genomes are tested by finding the combined scientific return V_{sum} of the n routes described by the genomes. This process is repeated until all genomes have been drawn at least k times. Once completed, each genome g is given a fitness value

$$F_g = \frac{\sum_{i=0}^{k_g} V_{\text{sum,i}}}{k_g}$$, where:

- k_g, $k_g \geq k$ is the number of times g has been drawn, and
- $\sum_{i=0}^{k_g} V_{\text{sum,i}}$ is the combined scientific return of all k_g tests.

Stigmergy

Section 1.1 explains why Full Communication is not realistic due to the excessive overhead. Stigmergy (see section 3.5) reduces the level of communication while still allowing some information to pass between the spacecraft. For the purposes of this thesis we have chosen to modify Tripp and Palmer's stigmergy method slightly, since it has already been successfully tested for space applications.

Our variation on Tripp and Palmer's stigmergy method uses periodic updates to modify the values of the asteroids and asteroid classes. Additionally, a number is assigned to each asteroid informing the spacecraft how many spacecraft plan to visit it. The spacecraft have no other information available regarding the other members of the swarm. They can not cross reference their plans with the others and collectively evolve good solutions. Instead, they must maintain a separate genetic population and attempt to use their limited information to guide evolution towards good routes that do not overlap with those of others.

There are four characteristics in the stigmergy method: greedy, considerate, proactive and obstinate. The greedy characteristic is simple; the agents want to maximise their fitness. Depending on how much the agent focuses on greedy behaviour, it may or may not sacrifice personal gain for the good of the swarm.

Our model of the asteroid belt has the considerate behaviour baked into it by giving zero fitness to the second BEE to explore an asteroid and reducing the value of a class based on how many examples of it has been investigated. Our reasoning for enforcing such a heavy focus on the considerate behaviour is that we have very limited resources, so we want to avoid overlap at all costs. By contrast, resources were renewable in the scenario used by Tripp and Palmer.

The proactive behaviour uses the information about the other BEEs in the swarm to help build a better global plan. If at least one other BEE plans to visit the same asteroid, the proactive fitness gain from that asteroid is reduced to zero. Agents which have a high focus on proactive behaviour will then likely change to another plan. The worst case scenario is that no agents visit the asteroid at all. But, as explained above, we are far more worried about overlap than we are about any particular asteroid going unexplored.

The obstinate behaviour provides extra fitness to asteroids which were part of the plan prior to the update. These are the asteroids that form part of the aggregate information sent to the swarm. This

added fitness is based on the scientific return of the asteroids in the route, so an obstinate agent with a good route is less likely to change its plans. The worst case scenario for obstinate agents is the opposite of the proactive one, namely that two agents have the same asteroid in their route. This can only really be resolved if there is a stigmergy update in-between the visits, which, thanks to our absolute focus on considerate behaviour, would cause the second agent to receive no fitness for planning on visiting the asteroid at all.

This leads us to the following fitness metric for spacecraft using the stigmergy method:

$$F_g = \sum_{a \in r} (V_a + V_c) \cdot (1 \cdot G + \{0,1\} \cdot P + \{0,1\} \cdot O)$$, where:

- $a \in r$ is the set of asteroids in the route up for evaluation;
- $V_a + V_c$ is the total scientific value gained by visiting the asteroid, as described in section 4.3;
- *G*, *P*, and *O* are the spacecraft's values for the greedy, proactive, and obstinate characteristics, which range from zero to one.

From this, one can see that the fitness award for visiting an asteroid ranges from 0% to 300% of the value in the model. The greedy characteristic is always active, as high value targets should always be preferred over low value targets, with an importance of *G*. For each asteroid, the proactive characteristic is triggered (*1*) if no other spacecraft plan to visit the asteroid, while it otherwise provides no fitness boost. The obstinate characteristic acts similarly. It provides a fitness boost governed by *O* if the asteroid can be found in spacecraft's previous plan.

No Communication

The last fitness metric needed for our experiments is the one used when no information is available at all. The spacecraft have no knowledge of the whereabouts, history or plans of the other spacecraft. Without any coordination between the spacecraft, heterogeneity is essential to avoid overlapping plans. Unfortunately, the personality characteristics found in the stigmergy method are not relevant here, as the spacecraft do not receive any updates.

This means we do not have the luxury of reusing the characteristics above. Instead, we have to be creative and devise potential characteristics. They must then be tested to see which ones increase overall swarm efficiency. These characteristics should reduce overlap by differentiating the spacecraft's decisions in a given situation. This must necessarily cause some spacecraft to choose asteroids which seem sub-optimal based on their available information. The selected characteristics should differentiate the spacecraft's decisions sufficiently to avoid overlap, but not cause a loss in spacecraft efficiency greater than the gain in swarm efficiency.

The point is, as stated, to differentiate behaviour. However, as the spacecraft spread out the probability of decisions overlapping approaches zero. The conclusion is, therefore, that the characteristics should make the spacecraft deviate from the perceived optimal choice enough that they diverge completely, after which overlap is no longer as large an issue.

The one stigmergy characteristic we could reuse is greediness, as it is the only one which does not rely on communication. As with stigmergy, however, the greedy characteristic is only useful as a mediator. The important factor is how much relative focus the agent sets on making high value choices over whatever other characteristics we can devise.

Another characteristic is the differentiation between early and late transfers; whether the spacecraft prefers to linger in orbit around an asteroid or transfer immediately. In practice, we simply provide a fitness boost or penalty to routes which take a longer time to complete. Essentially, we are

arbitrarily dividing the potential routes into multiple categories (in this case early and late transits) to encourage diversity. Ideally, the high fitness solutions should be evenly distributed between the different categories so that no agent systematically targets sub-optimal routes.

A third characteristic is for the spacecraft to prefer low cost transfers over high cost transfers. The locally optimal greedy choice is the transfer which provides the greatest scientific return per propellant cost. Hence, varying how much priority the agent gives to optimising each of these factors should increase diversity without major reductions in total utility. In practice, we simply use the number of transfer windows (which inversely correlates to average transfer cost) in the planned route to provide a boost or penalty as above.

A fourth characteristic would make spacecraft prefer to visit asteroids of the class they first visited, or to prefer asteroids of different classes. If the spacecraft somehow visit different classes to begin with, this characteristic should make them continue the trend. On the other hand, if they begin by visiting the same class, they will continue visiting the same class over and over. So this characteristic only helps if the other characteristics already ensure divergence. Asteroids belonging to rare classes may also never receive any attention because the benefit is negated by the penalty of visiting one more class. Because of these disadvantages, we do not consider this a viable characteristic.

Whatever characteristics we choose, the fitness metric would look like the fitness metric of the stigmergy method:

$$F_g = \sum_{a \in r} \left\{ (V_a + V_c) \cdot \sum_{i=0}^{n} (c_i \cdot C_i) \right\}$$, where:

- each C_i, $0 \leq C_i \leq 1$ is the importance of the i-th characteristic, and
- each c_i is the value used as input for the characteristic.

As can be seen, the adjusted scientific value is multiplied by the sum of the contributions from the characteristics. The first characteristic differentiates between high and low value targets:

$c_g = 1$, as per the greedy behaviour in stigmergy.

The second and third characteristics similarly differentiate between the time of departure from an asteroid and the cost of transfer:

$c_t = t_a$, where t_a is the time of arrival at the last asteroid in the route.

$c_c = |r|$, where $|r|$ is the number of transfer windows in the route.

The end result is that the fitness of a genome is determined by the spacecraft's perceived utility of the route it encodes, not the actual utility given by the model.

4.5.3 Selection Method

As touched upon in section 3.4.1, different selection methods have different strengths and weaknesses. Its not always obvious which is the best for a given task. Luckily, it is not particularly difficult to implement selection algorithms, and experimentation is one of the best ways of determining which one to use. So we will implement a number of different options and see which one works the best.

Our goal is to find a selection method which doesn't cause a significant computational overhead, while still providing just the right amount of evolutionary pressure. We do not want too much pressure, which could cause premature convergence. Nor do we want too little, which may lead to sub-optimal results.

The simplest way of determining which selection method to use is to try a wide range of them, and use the one which provides the highest scientific return overall. If computing time becomes an issue, we can also compare how long each function takes, and use the best one which completes in a reasonable time-scale.

4.5.4 Crossover Process

One of the weaknesses of using a direct representation is that individuals need to have valid routes as their genome, which makes crossover and mutation more complicated. It is not a simple matter of combining two good genomes, because if the first genome is cut at an asteroid a_i, the second genome needs to be cut at a_i as well.

As a result we have had to design a customised selection process. We start by picking out all individuals that include our current position. Normally, this would be the HIVE, but in stigmergy it can theoretically be any asteroid. It is also possible that the BEE is currently en route to an asteroid, in which case we use that one as the origin. The fitness values of these individuals can be compared by ignoring any part of the route which comes before our position or after the BEE has run out of propellant.

This allows us to select the first parent by comparing these individuals and using traditional selection methods (as described in section 4.5.3 above). We will look at a number of different selection methods to see which one works best.

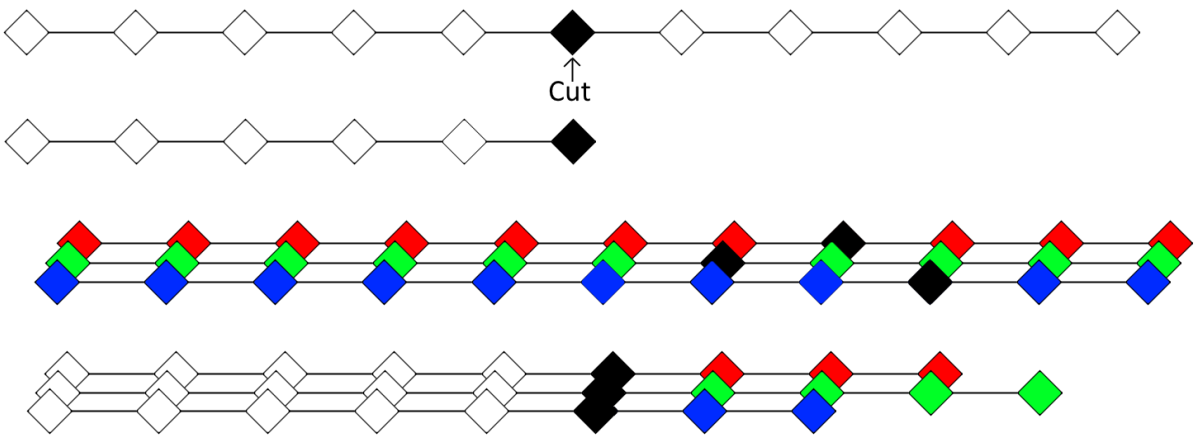

Figure 8: Shows the process for generating children. Once a fit genome has been found, a random crossover point is selected and all compatible genomes are found. A child is generated with each compatible genome by adding the remainder to the beginning of the first parent.

The other parent is less trivial to select. Our solution is to do things in a slightly different order. We generate a random crossover point before selecting the second parent (Figure 8). That leaves us with a single parent with a route that ends at an asteroid a_c. We can then search through the population to find other individuals which visit a_c at some point, and generate children by crossing these with our chosen parent. We naturally have to make sure that the second parent leaves a_c after the first arrives there.

We can calculate the fitness of these children, and compare them with children generated from other initial parents. While this potentially produces a large batch of children, the population can easily be cut down to size afterwards.

For mutations, the function is only slightly altered. Once a crossover point is chosen, we pick a random transfer window away from a_c, giving us one mutated gene. The route now leads to some other asteroid a_m instead, and we look for parents which visit a_m at the correct time. From there the

process continues as above. This means that all the children produced by this process would share the same mutated gene, which is a potential weakness of the method.

Another special case is when the crossover asteroid a_c appears very early in the first individual's genome or very late in the second individual's genome. It is possible that the route is shortened enough that the BEE still has sufficient propellant to transfer further at the end of the route. If this is the case, the route can be extended with randomly generated transfers (another form of mutation) or another crossover.

There are other ways to improve the selection process. We could go through each transfer in the route to find earlier or cheaper transfers; both of which would loosen restrictions for later generations. We could add individuals generated by other means to the population; greedy algorithms may provide decent solutions or partial solutions. Depending on how much time we have available, we may implement some of these improvements but our main focus is not on building an evolutionary planning algorithm, so time is a limiting factor.

Depending on our selection of parent asteroids and crossover points, we may end up with a lot of children or very few at all. So the question of maintaining some form of population control becomes relevant.

Population control looks different depending on which simulation is being run. With No Communication or Full Communication, the plan is generated before the simulation, and the evolutionary process is only run once. With Stigmergy, however, solutions are evolved and improved as new information becomes available.

With the pre-generated plans, we can generate our entire initial population by taking random transfer windows out from the HIVE. That way, our entire population consists of viable plans from the start. As our entire population consists of viable plans with known fitness, we can use traditional methods to eliminate the least fit members of the population.

With Stigmergy, things are a little more complex. When the first information update occurs, the BEEs have already implemented parts of their selected plan, so a lot of the non-selected plans are out of date; they assume the spacecraft took a transfer window it ignored.

We can divide this population into two classes. First, we have the genomes that represent viable plans. They include the BEE's current asteroid and present a complete route from there. The second class does not include the current asteroid, but may still contain good routes for later crossovers. Initially, the first class is going to be fairly small compared to the second, but it grows quickly as new solutions are added to the population.

Some members of the second class will have been used as parents for the next generation. It is possible that we could just remove these from the population, confident that at least some of their genes live on. This is not ideal, but neither is maintaining a large number of solutions with unknown fitness.

5 Implementation

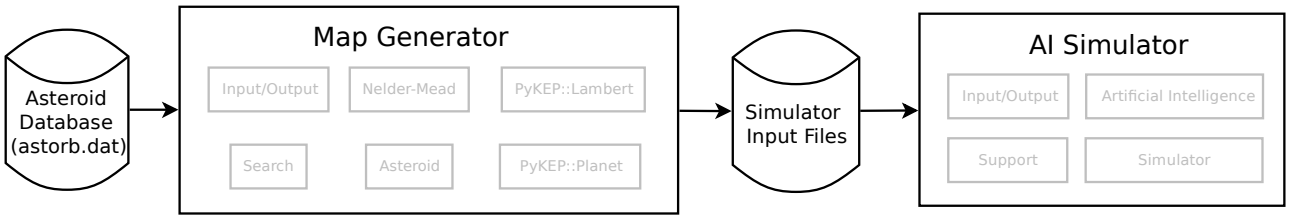

Figure 9: Architectural overview of the implementation. The Map Generator uses a database of all asteroids in the belt to produce a transfer map. The map is used as input for the AI Simulator, which uses the precomputed transfer windows.

To perform the experiments described in section 4.1, two main components were needed: a map of the available transfer windows and a simulation of the mission itself. It was natural to divide the parts into two separate code projects, as seen in Figure 9. The first project would compile a map of the asteroid belt transfer windows and store it in a file. The second project would then simulate the mission using the map and provide the desired experimental data.

This approach was chosen because of the inherent complexity of finding all available transfer windows between the massive number of asteroids in the main belt. We expected the program to take significant time to perform the calculations needed to generate the map; to the order of days, if not weeks. This approach would allow us to begin implementing the second part while the map was compiled.

5.1 Transfer Window Map

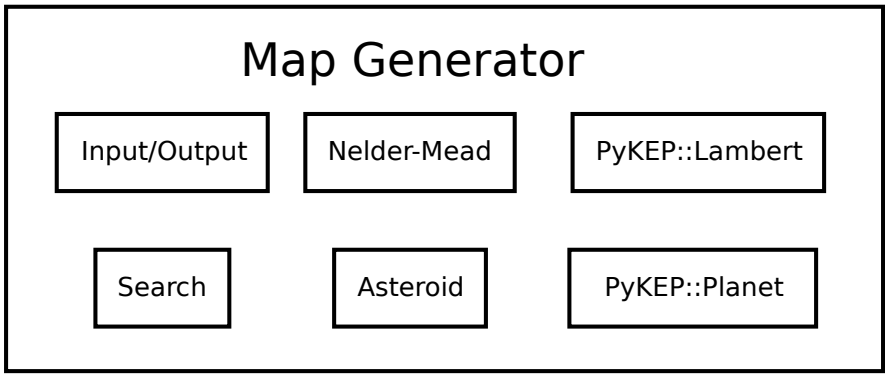

Figure 10: Major components of the Map Generator.

The overall plan for generating the database of transfer windows can be seen in Figure 10. After extracting asteroid data from the input file and generating asteroid objects, a depth first search is started at the HIVE's initial position. The search is recursively called on individual asteroids as they become reachable. The recursive call performs a one-to-all search for transfer windows using the Nelder-Mead technique to speed up the process. The PyKEP library (see section A.2) provides functionality for handling Keplerian elements and implements a solver for Lambert's problem. Once the search terminates, all transfer windows found during the search will have been stored to file.

5.1.1 The Depth First Search

A depth first search of possible transfers can be more readily thought of as a tree structure of possible routes. We would explore outwards from the HIVE along the first reachable asteroids in the database, backtracking and iterating through possibilities whenever we run out of propellant or

have explored all the possibilities from here on out.

As most asteroids could be reached through multiple routes, it was not strictly a tree search. We needed to keep track of which asteroids had already been explored to avoid duplication of effort. On the other hand, if a later route could reach an asteroid with more propellant to spare, it may be able to reach further than the last exploration had. Our solution was to keep track of the highest delta v available to spacecraft as they reached each asteroid. This information was propagated down the transfers out from asteroids, so that the whole network remained up to date on the costs.

The resulting method was as follows (see also Algorithm 1 below):

1. The delta v available to the spacecraft as they arrive at an asteroid is compared with the highest remaining delta v found so far. In the case of newly reached asteroids, this is 0m/s. The HIVE is a special case, where the delta v available represents a full tank of propellant. If the delta v available by the current route is higher than the highest found before, the available delta v is updated.
2. If we have have not already done so, we perform a one-to-all search of transfer windows out from this asteroid as described below. In addition to storing every transfer window, we also keep track of the cheapest transfer window to each target asteroid. An asteroid is reachable if the cheapest transfer to it costs less delta v than we have available.
3. If our available delta v has increased, either because this is our first visit to this asteroid or because we have found a cheaper route, we recursively call this method on our reachable asteroids. For these recursive calls the cost of the cheapest transfer to the reachable asteroid is subtracted from the available delta v at the origin asteroid.
4. Once all reachable asteroids have been updated and returned, the method returns.

Below is the same algorithm written in pseudocode:

```
All asteroids get an initial maxDeltaV of 0
The HIVE has the maxDeltaV of a full propellant tank
Starting at the HIVE, whenever an asteroid is reached:
    If it is reached with more delta v than the previous maxDeltaV:
        maxDeltaV is updated to the new delta v
        If we have not searched for transfers from here, do so
        For every asteroid we can reach:
            Check how much delta v we have left when we reach it
            Call this method on that asteroid with that delta v
```

Algorithm 1: Pseudocode implementation of transfer search algorithm.

5.1.2 Finding Transfer Windows

Transfer windows between asteroids are usually found by determining the position of local minima in pork-chop plots, as described in section 2.1.1. Evaluating every point in the plot using Lambert's Problem at a resolution high enough to be sure of having found the minima is computationally intensive. While it is certainly well within our capability for any one pair of asteroids, there are simply too many asteroids to allow such lavish use of computing power.

Instead we planned to use a gradient descent algorithm to quickly locate the minima. The main advantage of such methods is that they require few evaluations of the delta v cost compared to a full, high resolution evaluation of the whole search space.

Our search space was a 2 dimensional region spanning six years of departure times (as per the planned APIES mission duration), and up to six years of transfer times. While we suspected most transfers would be under one or two years in duration (about half the asteroid's orbital period is typical), it was possible that some transfer windows may feature a long, slow coast towards the target asteroid. So we could not rule out longer transfers.

The function of the pork-chop plot is the delta v cost of a transfer with a given time of departure and transfer time. This can be calculated in a straightforward manner if we have the velocity vectors of the BEE before and after each engine burn. The delta v cost of a burn is the change in velocity, so we simply calculate the magnitude of the difference between the final and initial velocity vectors to find the cost. As explained in section 2.1.1, we need two burns to complete a transfer. The cost of the two burns combined is the total delta v cost of the transfer.

As this function is quite complex, it is practically impossible for us to determine its derivative function. Many gradient descent algorithms rely on access to such a function, so we found ourselves with limited choice of which function to use. A well-known technique for these conditions is the Nelder-Mead technique, described in section 3.7. Nelder-Mead, in addition to being able to handle the problem at hand, is also known for being fast to converge in most cases and requiring few evaluations of the target function. This made Nelder-Mead an ideal fit for our purposes.

Each time Nelder-Mead is run, it finds one local minimum. To find all transfer windows between a pair of asteroids, the method has to be run multiple times. The result is a set of possibly overlapping local minima. Determining which elements of the set to keep and which were effectively duplicates was not trivial. At this point, we had very little information about how short transfer windows usually are in the asteroid belt. Without that information, it would be difficult to determine whether two discovered “transfer windows” represented two separate valleys in the pork-chop plot or were both in the same valley.

However, whether the elements were clustered around the same transfer window or represented several nearby windows did not really concern us. If the windows were close enough to each other, both in in time of departure and duration of flight, the BEEs would always prefer the one with the lowest delta v cost. Similarly, if there were both long and short transfer time options at around the same departure time and the shorter transfer time was cheaper, there would be nothing to gain from taking the longer transfer. Which left us with a much simpler problem. We could simply compare each element of the set with the other elements, and eliminate those which fit either of the following criteria:

- There exists an element in the set with a lower delta v cost and a shorter duration of flight, and with a time of departure within a given threshold value, or;
- There exists an element in the set with a lower delta v cost, and with both time of departure and time of flight within a given threshold distance.

Picking the actual threshold value would have to be done by experimentation. We eventually settled on 10 days, finding it a good compromise between reducing the complexity and giving the BEEs enough options.

5.2 Implementation process

The base functionality for performing the search and generating the file was completed as planned and expected. Implementing the Transfer Map Generator took far more time and effort than was planned for, however. The process of implementing the Nelder-Mead search and solving Lambert's Problem can be found in this section, while a description of the problems we faced and the process of solving them can be found in Appendices B, C, D, and E.

5.2.1 Lambert's problem

Lambert's problem, as described in section 2.1.1, is the challenge of finding transfer orbits between two given points at a given time. While techniques for solving the problem has been known for centuries, they tend to be fairly complex mathematical operations. Luckily, we did not have to

devote time and resources to implement any of them ourselves, as the ACT's PyKEP library (see section A.2) comes with an excellent solver built-in.

The PyKEP library proved easy to work with, even if we spent most of the first day hunting down a mysterious bug (see section H.1). The built-in Planet class had everything we needed to translate between the Keplerian elements used by our database of asteroids and their position at any given time. A built-in method of the Planet class takes in a time and returns the position and velocity of the celestial object it represents at that time. We could insert that directly into Lambert's problem, using the time of departure and the time of arrival as parameters for the two asteroids in question.

The Lambert's problem solver returns the transfer velocity both at the departure point and, crucially, at the arrival point. The delta v required for the first burn is the difference in the velocity vectors of the departure asteroid and the transfer orbit at the departure point. For the second burn, we would have had to propagate the spacecraft's motion along the orbit to determine its velocity at the start of the second burn. But, to our pleasant surprise, PyKEP proactively provided the vector we needed.

The first running version of our code calculated transfer costs between Earth and Mars, and was able to replicate pork-chop plots generated by NASA (Figure 11 below). Our delta v costs were off by a significant margin, as our code does not take the planets' gravity wells into account. A significant amount of delta v is expended in counteracting the force of gravity when leaving the Earth, unlike the asteroids our code is set up to handle. As a result our windows' departure and arrival dates were shifted towards each other to some degree. The topmost window has shifted downwards and left, while the bottom window has shifted upwards and right by about two weeks.

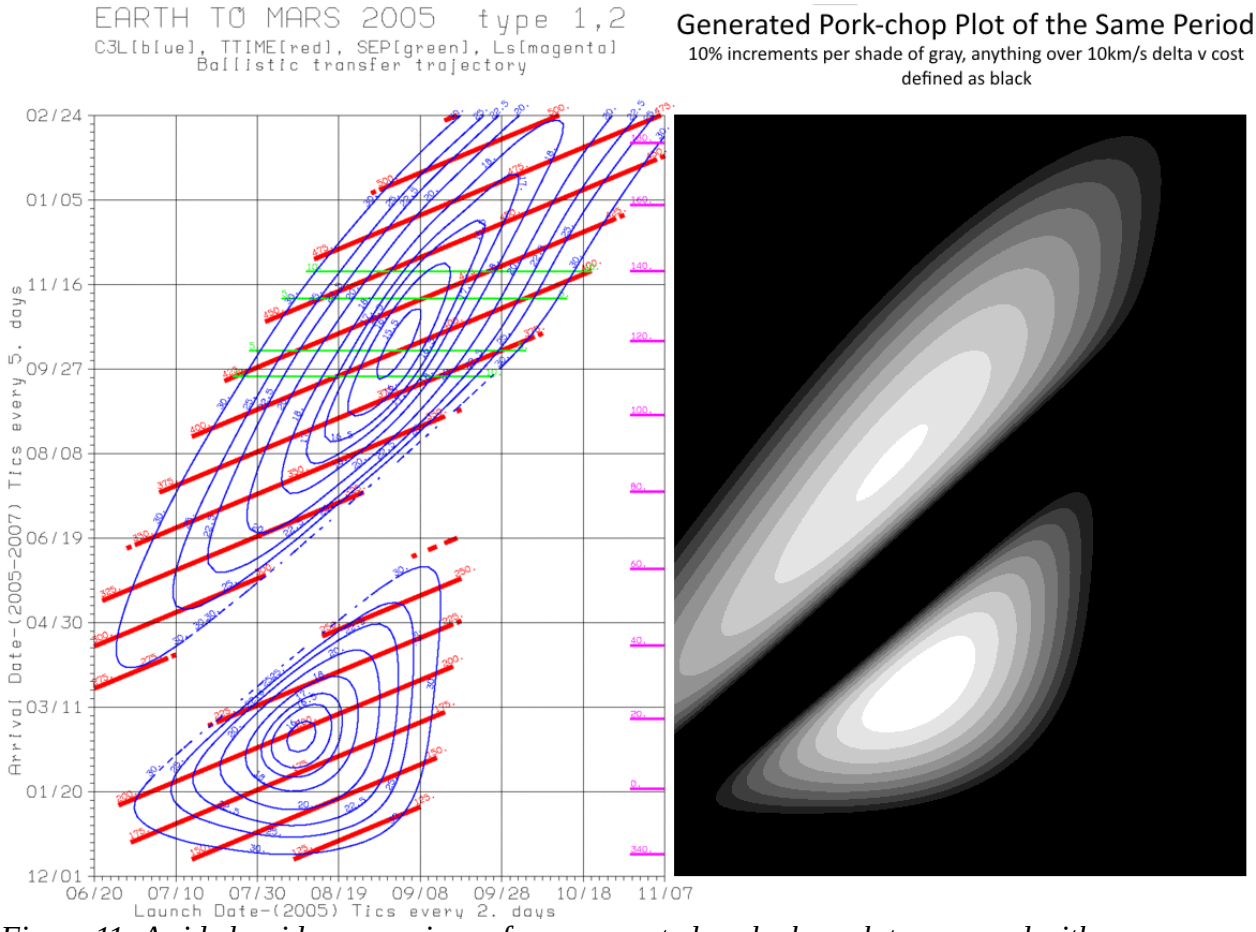

Figure 11: A side by side comparison of our generated pork-chop plot compared with one generated by NASA for the Mars Reconnaissance Orbiter.[45]

Each asteroid in our code is represented by an instance of the Asteroid class. Each Asteroid has a

Planet class from the PyKEP library to handle the orbital parameters, a list of the cheapest transfer window to all other asteroids, and a few other values required by the search algorithm. The list of cheapest transfers is used to determine which of the destination asteroids need to calculate their own transfers. Whenever the available delta v is increased (by a cheaper route to this asteroid being found), we can iterate through that sorted list, propagating the increased value to each one in turn. If any of these need to make updates of their own, the code is paused until the destination asteroid is finished, as is normal for depth first searches.

5.2.2 Nelder-Mead

The Nelder-Mead components were not quite as easy to set up. We tried a number of different implementations[35–39] over the course of a few days. None of them proved satisfactory. Issues included outdated code, poorly documented installation instructions, poor documentation in general, code which would not compile out of the box, and in one case a price tag of 10 pence per line of code. Eventually, we decided that it would be easier to implement the algorithm on our own.

The implementation process took a day, all told, and most of that was integration and debugging. Actually coding up the algorithm took a little more than an hour. While the first write up of the code worked correctly, it was very hard to read or analyse, so we made the decision to spend a bit more time on producing helper methods.

We produced a Flight class, which would contain all the information about a given transfer and handle the mathematics required by Nelder-Mead. Most importantly, we implemented comparators, so that we could compare two flights directly rather than having to extract the delta v costs each time. This also made sorting the flights in order of propellant efficiency easier. We also included the addition and multiplication functions used by Nelder-Mead to move vertices around, so that we could use an instance of the Flight class for each of our vertices.

As the most expensive to calculate part of a Flight is the delta v cost, we implemented a lazy evaluation of it. Around half the Flights created by an iteration of Nelder-Mead are never evaluated, so we managed to preemptively halve our run time.

Another issue which had to be settled was determining what initial simplexes to use, and how many to evaluate in order to ensure that all the transfer windows were located. Our initial solution was cover the solution space in simplexes, ensuring that every minima was located within a simplex. We divided the solution space into a grid, and divided each cell in the grid along the diagonal, creating two triangular simplexes for each cell. As detailed in section B.2, this turned out to be unnecessary.

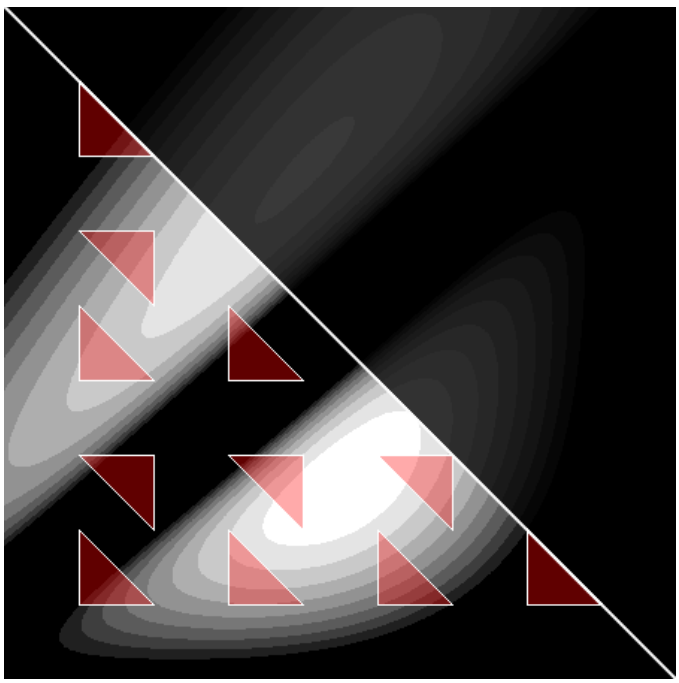

Figure 12: Example illustrating how the Nelder-Mead simplexes were placed in the search space. The pork-chop plot shows the cost of transfer from Earth to Mars in 2005.

Our initial grid resolution for the placement of Nelder-Mead simplexes was 1000x1000, which meant roughly two days between each vertex. The idea, supported by some of our initial optimisations (see Appendix B), was that most of these would be excluded and that

only the simplexes near the minima would be examined. Throughout the optimisation process (ibid), the resolution was repeatedly reduced. Figure 12 illustrates the final placement of simplexes in our first search.

5.3 Assignment of Asteroid Values and Classes

In addition to a map of transfer windows, the AI Simulator also needed the scientific value of each asteroid. There were two strategies: generating them at runtime, and generating values once and storing them in a file.

We opted for the second strategy to ensure that results are reproducible and to ease analysis. Additionally, as described in section 2.3, the class of most asteroids remain uncertain. Because our evolutionary algorithm relied on the scientific value gained from exploring multiple classes, those without an officially designated class needed to be assigned one by us.

The process involved loading a database of asteroids, assigning a random value (based on the distribution described in section 4.1.1) to each, and storing it in a new file with asteroid name, value, and class. Asteroids without a class were assigned classes at random, with a probability distribution equal to the distribution of the main classes in the mission area (section 2.3), depicted in Figure 13. To differentiate between asteroids with classes assigned by us and those with known classes, the assigned classes were prefixed with "ASS_". The algorithm was implemented in a short Python script, seen in Algorithm 2.

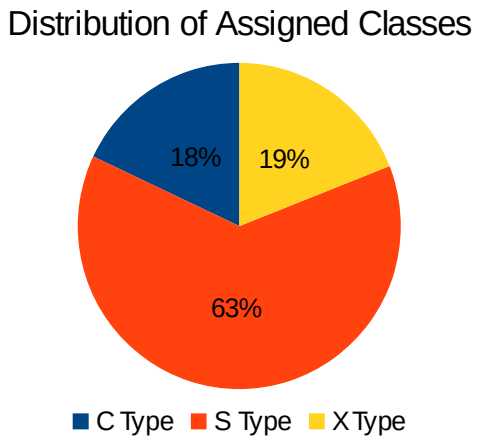

Figure 13: The distribution of randomly assigned classes from the three main types of asteroid in the mission region.

```
For each entry in the asteroid database:
    Retrieve its name and classification
    If the classification is uncertain (contains a question mark):
        Remove the question mark
    Otherwise, if the classification is unknown:
        Assign a classification at random
    Set asteroid value to a random number between min value and max value
    Write the modified entry (<name, classification, value>) to output file
```

Algorithm 2: Pseudocode explanation of how asteroids are assigned classifications and values.

5.4 Revising the APIES Plan

As we started testing our program on smaller subsets of the Transfer Map, we quickly discovered that transfer costs in the asteroid belt were higher than expected. In a map of 2095 asteroids, we found over seven thousand transfer opportunities with a cost under 1700m/s, but only twenty with a transfer cost under 500m/s. As the APIES BEEs only have 1700m/s of delta v available in total, this meant that each BEE would only get to explore at most 2-3 asteroids. Of even greater concern was that there were only three transfers out from the HIVE which cost less than half of the BEE's available delta v, and all three were to the same asteroid (2008 UK144; which happens to have a very similar orbit to the HIVE).

While we expected this to be less of an issue when the full Transfer Map of asteroids was used, rather than a randomly selected subset, it did suggest that the BEEs as envisioned by the APIES plan were not optimally suited for this sort of exploration. The individual BEEs had relatively little

propellant, as they were designed to make relatively small manoeuvres to intercept whichever asteroids happened to pass through their dragnet, rather than actively seeking out asteroids on their own.

The APIES plan (detailed in section 2.2) calls for exactly 19 BEEs in order to create a complete hexagonal drag-net (as seen in Figure 5 on page 8) centred on the HIVE. Available delta v per BEE is dependent on how much of the Soyuz-FG/Fregat's payload is spent on propellant rather than additional spacecraft. Since we do not need exactly 19 BEEs, it may be worth while to optimise the mission load-out towards maximising the total available delta v.

Table 4 shows the relationship between total mission delta v and the number of BEEs used. The APIES mission description has a detailed breakdown of the mass usage on the BEEs, but bundles the propellant tank mass together with other structural mass. This makes it difficult to determine how adding propellant effects the total mass of the BEEs. Thus, we have made the simplifying and pessimistic assumption that the structural mass scales linearly with the mass of propellant carried. Structural mass does include the propellant tank, whose mass would increase in tandem with its volume, but the relationship is sub-linear and a substantial fraction of structural mass is entirely independent of propellant mass. Our assumptions are therefore slightly pessimistic with regards to available delta v per BEE.

A simple analyses of the APIES mission load-out suggests that bringing 12 BEEs and using the extra mass to carry more propellant per remaining BEE would maximise the total delta v available. That would leave each BEE with just over 3400m/s delta v, which should be enough to allow them to explore around 4-5 asteroids each.

Total Mission Delta V

Number of BEEs	*Mass per BEE*	Delta v *per BEE*	*Total mission* delta v
19	43.39kg	1,699m/s	32,284m/s
18	45.80kg	1,916m/s	34,501m/s
17	48.49kg	2,143m/s	36,424m/s
16	51.53kg	2,378m/s	38,040m/s
15	54.96kg	2,622m/s	39,333m/s
14	58.89kg	2,877m/s	40,284m/s
13	63.42kg	3,144m/s	40,876m/s
12	68.70kg	3,424m/s	41,086m/s
11	74.95kg	3,717m/s	40,891m/s
10	82.44kg	4,026m/s	40,262m/s
9	91.60kg	4,352m/s	39,170m/s
8	103.05kg	4,697m/s	37,578m/s
7	117.77kg	5,064m/s	35,447m/s
6	137.40kg	5,455m/s	32,729m/s
5	164.88kg	5,874m/s	29,368m/s
4	206.10kg	6,325m/s	25,298m/s
3	274.80kg	6,813m/s	20,439m/s
2	412.21kg	7,345m/s	14,691m/s
1	824.41kg	7,931m/s	7,921m/s

Table 4: The relationship between number of BEEs and total mission delta v.

5.5 AI Simulator

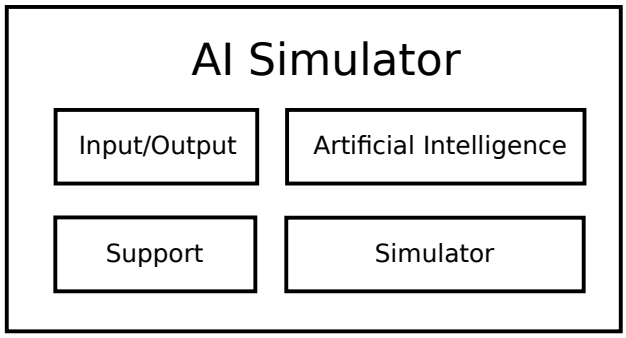

Figure 14: Shows the major components of the AI Simulator: the I/O component implements loading of the Transfer Map and asteroid values, the support code handles multi-threading and experimental set-up, the Simulator executes and evaluates plans, and the AI performs the evolution.

The main components of the AI Simulator are shown in Figure 14. It uses the Transfer Map to evolve and evaluate plans for exploring the asteroid belt. Its largest component is an Evolutionary Algorithm designed to produce plans with a high expected scientific return for the simulator to evaluate.

5.5.1 Plan

The overarching plan for the AI Simulator is described in chapter 4. The project consists of a model of the asteroid belt (section 4.3), a light-weight simulation (section 4.4), an evolutionary planner (section 4.5), and some surrounding framework. Each of these components would need further refining for each of the four experiments we plan to use them to run (see section 4.1). What follows are the implementation relevant details of the plan.

Input and Transfer Map Handling

The AI Simulator needed two files for input. The first was the file containing the asteroids, as per section 5.3. It contained asteroid names, classes and values. The second was the Transfer Map generated previously, with a file format described in Appendix C.

The two files would have to be stitched together to provide a data structure of the asteroids and the transfer opportunities between them, as seen in Figure 15. The transfer map would be built like a tree structure, with each asteroid object keeping track of its own transfer opportunities and each transfer keeping a reference to both the origin and the destination asteroid. The transfers out stored as a map in each asteroid, with the destination asteroid as the key.

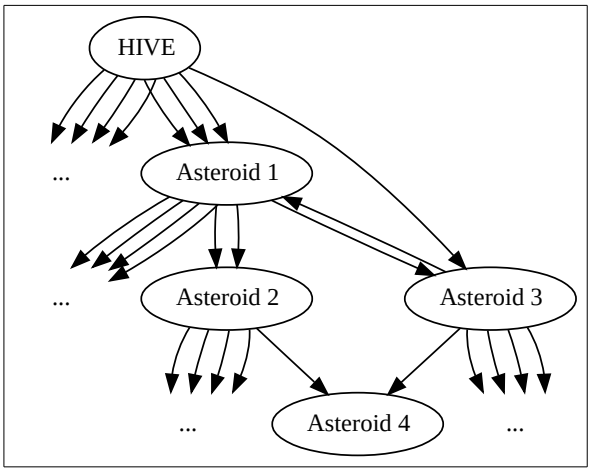

Figure 15: A visual representation of the Transfer Map. Each arrow represents a transfer window, which has a given time of departure, time of arrival and delta v cost.

An important thing to note is that the asteroids should not be changed in any way after being created. As the same transfer map would be used over every experiment, it was paramount that the experiments not alter any of the asteroid's traits. Which in turn meant that the simulator would need to keep track of which asteroids had been visited, rather than the asteroids themselves doing so.

The transfer window class would be very light-weight; little more than a (*C*-style) struct. Each needed a reference to two asteroids (origin and destination), a pair of time stamps (departure and arrival time), and a delta v cost. These values would be set by the input module as they were read in, but otherwise remain unchanged throughout the program.

Simulator

What all three experiments had in common was a need for a light-weight simulator to resolve the events of the mission. As the evolution for the Full Communication scenario needed to run this simulation multiple times for each genome evaluated, it would be preferable if this component was as efficient as we could make it.

```
For every route delivered to the simulator
     For each transfer in that route
          // Stigmergy Sims run with limited a time span at a time
          Check whether the transfer arrives within the time span to be simulated
               If so, create an event with that asteroid and arrival time
               Place the event into a list of events
Sort the event list in order of ascending arrival time
Iterate through the events in chronological order
     Add the event's asteroid to the list of visited asteroids
     If no asteroids of this class have been visited before
          Add the class to the list of visited classes
     Add one to the number of visits to the class
     // Override the following method to turn this into a fitness calculator
     Calculate the points earned from the event
     Add the points earned to the total score
Return the total score of the simulation
```

Algorithm 3: Pseudocode description of the simulator, including special considerations required.

The resulting algorithm (as seen in Algorithm 3, above) is deceptively simple. It takes in a list of routes from the spacecraft involved in the simulation, and transforms each route into a list of asteroid arrival events. These events are combined into one long list, which is sorted by the time of arrival at each asteroid. Then it simply iterates through this list of events, resolving each in rapid succession.

A few complications and special considerations do apply. First of all, the above algorithm did not evaluate the validity of the route, including whether the spacecraft had enough delta v to complete the route. This was done deliberately in order to keep the simulator scenario-agnostic. Responsibility for evaluating validity of the routes was given instead to the evolutionary algorithm and the individual spacecraft.

Secondly, the Stigmergy scenario involved running simulations of shorter periods in between each stigmergy update. This meant that we needed to be able to specify the starting and end times of the simulation, and needed to keep track of information about what asteroids had been visited across multiple partial simulations.

We also wanted to code to be general enough that a sub-class could be used to evaluate the fitness function of the No Communication and the Stigmergy genomes. This was done by simply performing the evaluation of the scientific return from an event in a separate method, which these sub-classes could later override.

Evolutionary Algorithm

The plan for the Artificial Intelligence is described in general terms in section 4.5. The genome would be a route which may be longer than it is physically possible for the BEEs to travel or exactly long enough, but never shorter. Should a genome end up being too short due to a crossover event or otherwise, transfers would be randomly tacked on to the end until no more are possible.

We wanted to avoid genomes shortening, as the utility of the mission was based on the total number of asteroids explored. Not exploring another asteroid when one is available would be wasting the remaining resources of the BEE. Additionally, the way our crossover algorithm was set up (see section 4.5.4) meant that the later transfers would experience more evolutionary pressure, so random extensions where possible was a useful way to prevent premature convergence.

The evolutionary algorithm selected all possible mates from a given genome and crossover point. This was done because of the relative rarity of viable mates, which would have to leave a given asteroid after a given time. As we did not know in advance exactly how rare viable mates would be, a method was implemented which made every possible crossover for a given genome, which could be used if it became difficult to find a mate in the population.

By the same token, we chose to use generational mixing. The children of the current generation competed with the existing population for the limited slots available for the next iteration. This allowed us to maintain a constant population size, despite a variable amount of children per generation. This also meant less evolutionary pressure when there were few new possibilities, and more pressure when there were many new options.

As described in section 4.5.3, we chose to implement a number of different selection methods. These would all be unexacting algorithms to code, and should not present any difficulties. The rest of the code was designed to be selector agnostic, using an interface to connect to any selector we had implemented.

Support Code

There were two main areas of support code. The first was the entry point of the program which would set up the experiments. It would perform the necessary initialisations and define what experiments to run and how many times to run them. The second part of the support code would handle multi-threading.

A configuration file, in the form of a static class, would contain all the system constants in order to ensure that things would remain consistent throughout our experiments. This file also would have details about how many experiments of what type to run.

The multi-threading component would use this information to schedule the experiments for an executor service, which would execute the experiments whenever a thread became available. The executor would typically have one thread running for each computer core, but the number could be overridden if needed.

5.5.2 Implementation Process

Unlike implementation of the Map Generator, the AI Simulator was implemented without much complication. There were some issues regarding Java's handling of the binary Transfer Map file produced by the Map Generator, described in section E.2. The plans for both the AI and the Simulator also had to be altered slightly to overcome programming limitations and unforeseen complications.

Simulator

The simulator was straightforward to implement. As we had intended, the choice of an event based simulation meant it was very easy to put the simulator together. All the complicated physics which usually makes simulators difficult to code had already been handled by the Transfer Map generator.

The biggest change from the initial plan came about from a desire to have more information available for debugging and the complications involved in parallel processing. In the original plan,

the simulator returned only a double value representing the scientific return of the mission. We wanted to have more information available, so we modified the code to return the simulator itself instead.

The simulator already had access to the routes of each spacecraft, which asteroids had been visited, which asteroid classes had been visited (and how many from each class), and just about anything else we might want to know about the result. As such, the modifications needed to turn the simulator into a results object were minor. A few extra get-functions were needed to be able to extract the required information and the code had to be modified to keep the routes in memory rather than just using them in the constructor.

The sub-classes which were to serve as fitness functions for the Stigmergy (Exp 3) and No Communication (Exp 4) scenarios were also easy to write. The code was general enough to handle varying numbers of BEEs, so could handle the fitness calculation for a single spacecraft. As planned, the calculation of scientific return for an event was placed in a separate method which the sub-classes could overwrite.

Of the two, Stigmergy was the more complicated. Because the simulator's environmental data was used as the available stigmergy information, we could not allow the testing of a plan to alter the state of the simulation. This meant that the evaluation of a genome's fitness would have to be done separately from the usual event handling code, which was linked to the environmental data. We later discovered that we had been unsuccessful in doing so, as documented in section H.6.

Evolutionary Algorithm

One of the principle advantages of evolutionary algorithms is that they are very easy to implement, even for complex problems. In some cases, ours included, the genome required special considerations with some of the genetic operators, but the algorithm itself is fairly robust. The only real implementation difficulty we had with the evolutionary algorithm was that parts of it had to be rewritten to allow for the Full Communication scenario's co-evolution requirement. Some processes also had to be pulled out into external methods so that the sub-classes could override them.

The basic evolutionary algorithm, which was also used directly by the No Communication (Exp 4) version of the code is detailed below in Algorithm 4. The process was typical of evolutionary algorithms. The population is seeded by randomly generated individuals, after which the generational code is run repeatedly until the final result is ready.

```
// No Communication (basic) Evolution
Randomly generate initial population
For each generation:
     Select a number of genomes in the population to generate children from
     For each of those genomes:
          Perform crossover on it and every candidate from the population
     Determine the fitness of each child
     Add all children to the population
     Select a number of individuals from the population to be the next generation
```

Algorithm 4: The basic evolutionary algorithm, as used by the No Communication scenario.

In each generation, a number of the highest fitness genomes were selected for crossover. As potential crossover opportunities were few, every possible crossover was performed. The fitness of each new genome was ascertained. For the No Communication scenario, each BEE had its own fitness object; while the other experiments used different methods to determine fitness. Once their fitness was known, the children were added to the population which was then pruned down to its original size.

As seen in Algorithm 5 below, the Stigmergy scenario (Exp 3) required two major changes from the

basic evolutionary algorithm. The first was that the evolutionary process took place multiple times as updates from the environment changed the expected fitness of current plans. After each stigmergy update, many members of the population would be outdated; they relied on the spacecraft having made different choices than it ended up doing. These plans were kept in a separate list in the hope that they would be useful for later crossovers.

```
// Stigmergy Evolution
Update the fitness calculator with the new stigmergy information
Copy the whole population into an array of outdated plans
Find every genome which leaves our current position after the current time:
    Those genomes are now our population
For each generation:
    Select a number of genomes from the population to generate children from
    For each of those genomes:
        Perform crossover on it and every candidate from the population
        Perform crossover on it and every candidate from the outdated plans
    Determine the fitness of each child based on available stigmergy information
    Add all children to the population
    Select a number of individuals from the population to be the next generation
```

Algorithm 5: The stigmergy evolution method, with differences from the basic process (Algorithm 4) highlighted.

The second change was that the spacecraft in the swarm shared a fitness object, which kept track of the available information. As with the No Communication scenario, this was a subclass of the simulator code. In every stigmergy update, this fitness calculator was updated with information about what the spacecraft had done and what they were planing to do.

With Full Communication (Exp 2), the whole evolutionary process had to be paused after each generation so that multiple simulations could be run to examine the fitness of each new individual. This co-evolution meant that the BEEs had to wait until everyone was finished with crossovers for a given generation before it could select who to include in the next, as the fitness depends on how well it worked with the other BEEs.

As seen in Algorithm 6 below, the Full Communication code overrode every part of the main evolution method. Rather than running an evolution on its own, the co-evolution code controlled separate evolutions for each BEE. Whenever the evolutionary code would have performed an action, the co-evolution instead called that method on each BEE. The code started by initializing each member of the swarm, calling an unmodified initialization method on each swarm member. Each generation was split into a generating phase and a selection phase, with the fitness evaluation sandwiched in the middle.

```
// Full Communication Co-Evolution
For each BEE in the Swarm:
    Randomly generate initial population
For each generation:
    For each BEE in the Swarm:
        Select a number of genomes from the population to generate children from
        For each of those genomes:
            Perform crossover on it and every candidate from the population
    While there is a child which has not been in enough simulations:
        Run a simulation with a random set of children which need more simulations
        Add the score earned to each child's list of scores
    For each BEE in the Swarm:
        Add all children to the population
        Select individuals from the population to be the next generation
```

Algorithm 6: The co-evolution method, with differences from the basic process (Algorithm 4) highlighted.

New No Communication Fitness Characteristics

While some concepts for communication-less fitness characteristics are discussed in section 4.5.1, we had not yet settled on exactly which to use when we started implementing.

We considered using the scientific return of the route directly as its fitness. If the search area was wide enough, chances are each BEE would find a different route. If the area is not quite so wide, there would be issues with multiple BEEs picking the same route. In early testing, we found that with just the scientific return based greedy behaviour, we typically had one or two pairs of BEEs pick identical routes in each run, even in larger test sets. Clearly, more variation was needed.

The other two behaviours we had already considered were both based on the same principle. We would divide transfers into two categories, based on delta v cost or time of flight. If the time or delta v cost was below a given threshold, the transfer would receive a fitness boost. The boost would depend on how highly the spacecraft prioritised the behaviour in question and how valuable the target asteroid was.

During implementation, we hit upon a less binary solution, which would provide a proportion of the full fitness boost to transfers which were above the threshold. We knew, from cursory examinations of the transfer map, that there were significantly more high cost transfers in the map than low cost transfers. As a result, rewarding only low cost transfers would have little effect. If only 1% of the transfers were eligible for the fitness boost, chances were that there was a transfer in the remaining 99% which produced enough fitness to make up for the difference.

We could have picked a threshold which neatly bisected the search-space, but that would provide a static boost to a wide range of transfer costs. This would not provide much pressure towards reducing transfer costs. So the proportional approach was chosen. About 1% of the transfers in the map would get the full boost. However, about 40% would get half the boost or more, which gave us the best of both worlds.

The formula was simple. If the cost or transfer time was equal or below a chosen threshold, the transfer got the full fitness boost. If not, it got only a fraction defined by the threshold divided by the cost. So if the transfer cost twice the threshold, it got half the fitness boost.

One of the fitness characteristics we had quickly abandoned was the idea of having each BEE specialise in a given class of asteroid, preferring to keep visiting the same ones. However, given the nature of the solution space, we ended up implementing the reverse. As described in section 5.3, the vast majority of asteroids in our transfer map belong to one of three classifications (assigned randomly by us). As we expected each BEE to visit at least three targets, assigning a fitness boost to classes not yet visited should result in roughly an equal number of each class being visited. At the very least, we would expect each BEE to visit each class once with such a boost, meaning at least 12 of each would be visited.

This would not necessarily improve the diversity, in the sense that it did not prevent the BEEs from targeting the same asteroids, but it would increase the final fitness by ensuring that no class becomes significantly over-represented. It would neatly allow us to incentivise diverse sampling without requiring communication.

5.6 Analysing the Output

One of the first things we noticed when we started testing with a nearly complete data-set was that the routes were shorter than we would have hoped. With No Communication, the BEEs typically took routes with a length of 2 asteroids, while the Full Communication BEEs generally managed 3. This was discouraging, as we were averaging 30-40 asteroids explored; far less than the 100 envisioned by APIES. It also meant that our evolutionary algorithms were not particularly useful.

5.6.1 Selection Methods

As explained in section 4.5.3, we planned on implementing a number of different selection methods and seeing which of them worked the best for our purposes. Once the rest of the code was

reasonably stable, we ran some trials to see how the selection methods compare. We did 20 trials for each scenario for each selection method. The average scientific return and average number of asteroids explored are documented in Table 5, below.

Performance of Various Selection Methods

Selection Method	*No Communication*		*Stigmergy*		*Full Communication*	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Elitism	363.5	25.2	367.5	25.8	524.6	35.7
Roulette-wheel Selection	360.8	25.2	366.4	25.5	441.4	31.6
Roulette-wheel Selection, lowest fitness scaled to zero.	369.2	24.9	365.7	25.4	498.5	34.2
Tournament Selection, 3 genomes per tournament	366.7	25.3	361.6	25.4	489.8	34.3
Tournament Selection, 5 genomes per tournament	367.5	25.6	363.3	24.7	490.0	34.5
Stochastic Universal Sampling (SUS) Selection	348.4	24.1	351.5	24.6	489.9	33.9
SUS Selection, lowest fitness scaled to zero.	354.4	24.7	353.9	24.9	492.9	34.3

Table 5: Average scientific return and average number of asteroids explored for a variety of selection methods. 100 generations of 100 individuals, with 10 attempts to generate children per generation, a mutation rate of 5%, at least 4 co-evolution evaluations per full communication genome, and 12 stigmergy updates (one every 6 months).

The first and most obvious observation which could be made from this table was that the selection method has little effect on the utility of Stigmergy and No Communication runs. This may be because, unlike with Full Communication, these evolvers are not optimising towards the final score but towards an internal fitness value. The exact fine tuning of this fitness value had not yet been completed, so there may not be enough pressure towards achieving a higher score.

As described in section H.6, we later discovered another reason for the poor performance of the experiments with internal fitness representations. Due to a bug in the calculation of class value, the evolvers were optimising only for the asteroids' intrinsic values. As a result, they were not particularly effective at picking good solutions, no matter the selection method.

The Full Communication scenario, which used the scientific return as the fitness directly, made a better comparator for the selection method. In this column, it can be seen that the scores came in three rough tiers. The unmodified Roulette-wheel Selection lay nearly 50 points below the next worst option. This was especially significant as the difference between the best and worst of 20 trials tended to be around 40 points. Similarly, Elitism was a full 25 points above the second best option, which was the high pressure version of Roulette-wheel Selection. The others were all within a 10 point range of each other.

It was telling, if not particularly surprising, that the two best selectors were the ones with the highest selection pressure. We knew that possible crossovers were rare in our populations, which in turn lead to relatively few children being generated in each generation. With generational mixing, this meant only a few genomes were eliminated per generation. As such, we wanted to make sure

that the worst solutions are lost. This logic also applied to the other scenarios once they had been fully optimised and bug fixed.

5.6.2 Excessive Cloning

There is no easy way to do crossover when there are low odds that any members of the population visit the same asteroid. As a result, we only really performed crossovers when there was a duplicate of the genome in the population, so most of our children were clones. In this environment, a high mutation rate was the only way to keep the population evolving. And, in fact, we found we got the best results when our mutation rate was about 60%.

Our first order of business was to prevent the premature converge caused by excessive cloning. As we had generational mixing, it would not be helpful to produce any copies of an existing member of the population. So we simply pruned out any children which were identical to one of their parents. This gave us better scientific return on runs with low mutation rates, but did not increase overall scientific return.

Before the cloning issue was resolved, mutation rates between 30-90% provided roughly the same utility, although we got the best results with about 60-70%. After the crossover code was rewritten to prevent cloning, we got significantly better results with low (0.5%) mutation rates. Mutation rates between 10-85% now provided roughly the same utility, higher than any lower rate but lower than without cloning measures.

While examining possible theories for why the routes remained so short, we implemented a simple method which pruned out routes of length one from the transfer map. It simply checked whether each transfer out from the HIVE arrived in time for any further transfers, and removed any which which did not. The idea was that by making sure that every route was at least two asteroids long, we could improve crossover and remove the least useful parts of the search area.

The process worked, as can be seen in Table 6, but not nearly as well as we could have hoped. There are consistent improvements in scientific return for every possible mutation rate. The results were still not any better than before we started making adjustments, however, and the routes were still not as long as we would have liked.

Effects of Varying the Mutation Rate

Mutation Rate	*Average (Before anti-cloning measures)*	*Average (After anti-cloning measures)*	*Average (anti-cloning and pruning dead ends from HIVE)*
100%	471.1	444.1	465.5
95%	518.6	485.5	511.5
90%	534.0	499.4	517.8
85%	533.1	512.9	528.3
80%	536.4	517.7	524.5
75%	538.0	516.4	525.5
70%	538.3	521.5	525.3
65%	535.3	523.4	529.0
60%	544.1	510.6	527.8
50%	540.8	521.5	531.6
40%	533.7	522.8	533.2
30%	532.5	522.1	528.2
20%	527.8	522.4	524.7
10%	508.0	522.0	525.9
5%	491.0	496.9	525.8
0.5%	451.9	508.5	513.8

Table 6: Average scientific return explored for a variety of mutation rates, using full communication. 100 generations of 100 individuals, with 10 attempts to generate children per generation, an Elitism selector, and at least 4 co-evolution evaluations per full communication genome.

5.7 Extending the Mission

We found that the BEEs typically ran out of time before they ran out of propellant. It wasn't uncommon to see a BEE with over 1000m/s of delta v left at mission end. We looked at when they arrived at the last asteroid in their route and discovered, with few exceptions, that every BEE arrived within a day of the end of the mission.

A quick examination of the transfer database showed that half of all possible transfers only arrived in the last few days of the mission. The reason for this was obvious in hindsight. The Nelder-Mead simplexes (see section 5.2.2) are limited to searching within the available time of the mission. As a result, whenever the search function is rising towards a peak with a time of arrival after the mission conclusion, Nelder-Mead finds a “peak” which arrives as late as possible. Additionally, while transfers which leave too early for travel are eliminated (see section E.1), later transfers are not, leading to late transfers being over-represented.

The result is that, without a lot of evolutionary pressure or a high mutation rate, it is very hard for the genomes to grow in length. If they are extended with an arbitrary transfer window, they will reach the next asteroid in the last few days of the mission. This would in part explain why we were getting so short routes.

While looking for other explanations, we did some statistics on the transfer windows we had thus far generated. The average delta v cost of a transfer was about 1000m/s and the average flight time was about 22 months. With 3400m/s of delta v available and 72 months worth of mission time, it seemed we were short on both.

In and of itself, the delta v should not be a problem. While only a quarter of the transfers in the database cost under 900m/s and only 5% cost under 650m/s, the evolutionary algorithm should be powerful enough to find these cheap transfers. The problem is that waiting for cheap transfers means less time for more transfers later, which means it is presently smarter for the BEE to make many early, expensive transfers than to wait for cheap ones.

We expected that if the mission was not arbitrarily terminated after 6 years, the BEEs would be capable of finding routes of length 5 or even 6 by taking their time and waiting for better options. Our current data-set did not cover that time-span, however, so we would have to generate a whole new Transfer Map. Increasing the mission duration would require an even more time consuming search. Since the last one had run for 49 days and not yet finished, we were naturally hesitant to begin anew.

However, as described in section E.1.1, we had in the mean time estimated that filtering on arrival time in addition to delta v would massively reduce the number of asteroids we would need to evaluate. With these new improvements, we estimated a five-fold reduction in run time, which would allow the seven weeks of execution to be replicated by a new one week search, and the entire search to be completed in another.

Increasing the mission length would unfortunately increase the problems size, and thus the time needed to complete the search. Whether the increased performance of a search algorithm would suffice for allowing the larger problem to be solved required more thorough estimates than could be made without actual implementations.

5.7.1 Adding Time Restrictions to the Search Algorithm

Up to this point, the search had restricted branching and depth of the search only by considering delta v costs of transfers. Whenever new transfer windows were found, they were used to improve the estimated cheapest cost of arrival for any asteroid reachable through them. Asteroids which could never be reached cheaply enough were never included in the search as reachable asteroids.

As seen in E.1.1, this could be improved by considering time. The search could be changed, as shown in Algorithm 7, to attempt to enumerate every possible route, by using estimates of both cheapest cost and earliest arrival to decide whether to make a recursive call. It would have two parameters relevant to the algorithm: time of arrival and available delta v. Only outgoing transfer windows with a lower delta v cost than available and a time of departure after the arrival would now be considered.

```
recursiveCall(timeOfArrival, remainingDeltaV):
    If outbound search has not been done:
        Search for outbound transfer windows
    // previous best initialised at end of time and negative delta v
    If timeOfArrival or remainingDeltaV improves previous best:
        Update previous best        // one or both values
        For all transfer windows:
            If timeOfDeparture > timeOfArrival & cost < remainingDeltaV:
                // make recursive call on destination asteroid
                recursiveCall(time of departure, remainingDeltaV – delta v cost)
```

Algorithm 7: The algorithm for restricting on time as well as delta v *in the search.*

At any node in the tree, it would generate outgoing transfer windows the first time it was reached, and attempt to reach other asteroids with the available time and delta v. As the search progressed,

fewer and fewer transfer windows would be available due to the ever decreasing amount of delta v and time available for outbound transfers. This would result in a smaller branching factor than the previous search, which only considered delta v.

To avoid making redundant recursive calls that cannot expand the search, every asteroid stores the highest amount of remaining delta v and the earliest arrival it has ever been called with. Figure 16 illustrates the situation. There is no point making new recursive calls if neither the highest delta v or earliest arrival is improved. An earlier arrival would allow windows with earlier times of departure to be traversed, while a higher amount of delta v remaining would allow transfer windows to be called with a better amount of delta v remaining. If neither is improved upon, there is no point making a recursive call. Because every transfer window requires a strictly positive amount of delta v and time, a search will never progress after arriving at an asteroid it has already visited. The search would arrive with both less delta v and less time available than before.

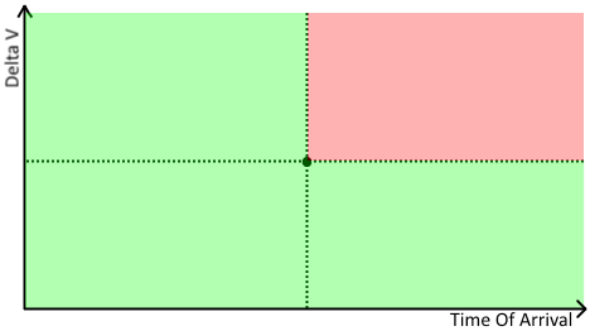

Figure 16: Illustrates the bounds check of the search. The two lines show the lowest delta v *cost and earliest arrival time seen by an asteroid. Only a recursive call that moves the intersect point towards the bottom right (green area) could affect the result of the search.*

The new and improved search algorithm was benchmarked as described in F.1. The speedup was measured at a factor of roughly 13-14, while the initial estimate that led us to alter the algorithm suggested a fraction of this. If the actual speedup would be equal to this measurement, the seven week search would be completed in just two to three days and the entire search likely within a week. The entire project would have been pushed ahead by weeks had the source of improvement been discovered at the implementation stage. The speedup would likely allow a second search to be performed if the longer mission would not increase the problem size several times over.

5.7.2 Expanding the Mission Length

As discussed above, the original 6 year time span of the mission limited the number of transfer the spacecraft could make. The BEEs typically had about a 1000m/s of delta v left at the end of the mission time, and were ignoring cheap transfers in favour of earlier options. Expanding the time span would allow them to select cheaper, but later transfer windows and still not run out of time.

Inspection of the Transfer Map revealed that no transfer windows had a duration of flight longer than 4.4 years out of the available 6, and only 0.3% above 4 years. The reason for this is that transfers much longer than this would require more than one revolution around sun, which would be less efficient than half revolution transfers.[40]

The choice of mission length fell on 20 years instead of the original 6. This was over three times the original length, which should provide the BEEs with enough time to wait for cheaper transfers. We wanted to make certain the BEEs were able to expend their reserve of propellant before running out of time. More than 20 years would likely be unnecessary and move the new mission description further from the original. Less than this could fail to alleviate the problem.

With a 20 year mission, allowing for transfer up to 20 years would only waste computational time. Because the low percentage of transfer windows with duration of flight above 4 years, the duration of flight was limited to a maximum of 4 years. This would eliminate a large portion of the search space and allow the use of fewer Nelder-Mead simplexes.

The changes to the Nelder Mead algorithm were minor. Figure 17 shows the modified search space with the Nelder Mead triangles. Reducing the number of Nelder Mead simplexes to 7x1 compared to the 11 used in the 6 year search would mean a substantial speedup per asteroid yet still provide an adequate coverage of the search space.

The modified search space was benchmarked with the improved search algorithm, detailed in section F.2. It would seem that the additional mission length negates the speedup gained from modifying the search algorithm and reducing the number of Nelder-Mead simplexes, meaning a full search could take as long as the previous one. Fortunately, the new search algorithm produces only a fraction of the junk data, meaning it would produce roughly an order of magnitude more valid routes given the same execution time.

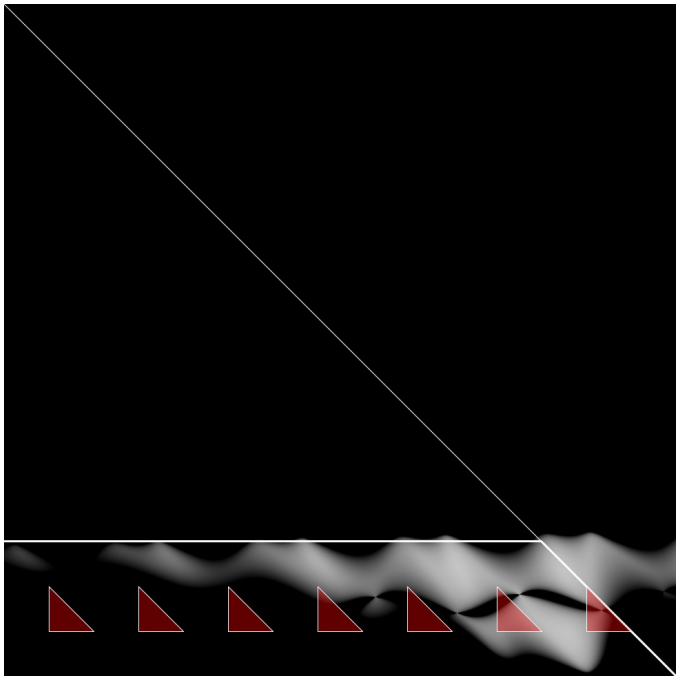

Figure 17: Shows an example of how the initial Nelder-Mead simplexes would be placed in the modified search space. The pork-chop plot shows transfer opportunities from 2006 SB102 to 2001 VX75 with a propellant limit of 1200m/s.

6 Experimental Setup

The experiments, described in section 4.1, have a multitude of parameters with values discussed and described throughout the previous chapters and some of the appendixes. This chapter summarises the parameters and the reasoning for selecting their particular values.

6.1 Transfer Map Parameters

The Transfer Map Generator requires many parameters to be set and tweaked before it produces the desired result. The parameters that are relevant to the search can be found in Table 7.

The mission start is defined in days relative to the reference epoch J2000, set at noon on the 1st of January 2000, Terrestrial Time. It has been set to the planned launch of APIES in 2020. As described in section 5.7, the mission was extended from 6 years to 20 years due to the short transfer lengths achieved with the shorter option.

The spacecraft are allowed a maximum of 3424m/s delta v. Section 5.4 describes why this has been increased from the original 1700m/s detailed in the APIES plan and how it allowed for better use of mission resources.

Transfer Map Generator Settings

Parameter	*Value*
Mission Start	7,305 days (1st of January 2020)
Mission Duration	7,305 days (20 years)
Maximum Transfer Duration	1,461 days (4 years)
Maximum Transfer Cost	1,200m/s
Spacecraft Delta V Limit	3,424m/s
Nelder-Mead Grid Resolution	15
Nelder-Mead Epsilon	1
Asteroid Database Size	212,077
Included Asteroid Percentage	100%

Table 7: Shows the parameters used when generating the Transfer Map.

Only transfer windows costing a maximum of 1200m/s are included in the search, as per section B.4. This was necessary to shorten the time needed to generate the Transfer Map, and has the side effect of ensuring that BEEs are not allowed to empty their propellant reserves in just one transfer.

The Nelder-Mead resolution has been reduced to the minimum of what would work reliably. Section 5.7.2 discusses the choice, which landed on 7x1 Nelder-Mead simplexes; 7 in the axis of mission duration and 1 in the axis of transfer duration. The resolution was sufficient for the search while not requiring more computational time than absolutely needed.

The precision of the Nelder-Mead algorithm, epsilon, was set to 1 day. This effectively meant that any two transfer windows with less than 2 days difference between them in both time of transfer and duration of transfer would be considered as examples of the same underlying transfer window. The one with the most expensive cost of transfer was eliminated. Section 2.1.1 shows an example where this precision was used by NASA, showing it is a reasonable choice. A Transfer Map where most transfers take at least a year would also not benefit from a relatively minute increase in precision.

The search used a filtered database, filtered down to about a third of the asteroids in the full database. Section B.3 details the process of eliminating asteroids based on estimated cost of reaching them, and why such a heavy reduction was necessary and justifiable given the computational cost of generating the Transfer Map. Benchmarking and testing used a reproducible subset of asteroids in the filtered database, based on what percentage of the problem size could easily be handled. For the full search, the percentage of asteroids to include was naturally set to the

entire filtered database.

6.2 Scientific Return Parameters

We initially considered using the priority information available in our asteroid database as a source for how much to value each asteroid. It turned out that most asteroids in the database had not been assigned any special priority, and that the priorities assigned were too astronomy specific to be much help to us. Most priority asteroids were given observational priorities because their orbits were not well known or there was a limited window for when to observe them, rather than any intrinsic scientific value from the asteroid itself.

So we had to produce our own asteroid values. The values for each asteroid were randomly generated from the set of real numbers between one and ten. The class values were assigned manually, however, with the values as seen in Table 8. The quick and dirty guideline we used was that classes with more than one letter were worth 30 points, and single letter designations were worth 20. This was done under the assumption that scientists would have some high priority target classes in mind in any such exploration of the Asteroid Belt. With no way of predicting what classes would be of special interest in the future, it might as well be these classes.

Reachable Asteroids

Asteroid Class	*Value*	*Frequency*
C	10	6
S	10	2
M	10	0
l	20	1
E	20	1
CP	30	1
ASS_C	20	7616
ASS_S	20	26608
ASS_X	20	7892

Table 8: Shows the assigned class values and number of asteroids of each in the reachable sub-set of asteroids.

The most common classes (C-type, S-type and M-type) were given a lower priority, at 10 points, as they are better understood than their more unusual brethren. Asteroids with classes assigned by us were given a slight bonus, since we assumed that any such assignment would be provincial at best, with a decent chance that a close examination would reveal misclassifications or other oddities.

Section 5.3 describes the process of assigning values and classes to the asteroids. Investigation of the asteroids included in the final Transfer Map showed that most of the asteroids were ones with assigned classes. This was unfortunate, as the likelihood of arriving at one of the 11 actually classified asteroids was extremely low. To make matters worse, three of the six classes with members in the final Transfer Map were only represented once, the other two twice and six times. As the class values do not reflect the relative rarity of these classes, the evolutionary algorithm would have little interest in them.

Had we known the relative distribution of classes in the final data set before they were assigned, we would have made sure the assignment method included all classes, not only the three most common ones. This would have boosted the number of asteroids not belonging to one of the main three classes significantly and made them more important for the evolutionary algorithms. The class values would also have been adjusted to reflect the extreme difference in frequency. The problem was unfortunately not detected in time, however.

6.3 Baseline: Monte Carlo Simulation

For the Monte Carlo simulation the main determining factor in how well the simulation represents the solution space is how many random solutions are generated. The more solutions we could test, the better they would represent the solution-space. On the other hand, both time and computing power were limited.

Each BEE in a Monte Carlo simulation makes arbitrarily chosen transfers from its initial location, with an equal probability of picking each transfer. As long as the BEE has the time and propellant to make a transfer, it will keep picking random ones. The output of a single Monte Carlo run is the combined scientific return from all 12 BEEs in the random mission.

One million (1,000,000) routes were generated by the Monte Carlo Simulation to ensure a smooth distribution of results. We expect the scientific return to range in the hundreds, which would mean thousands of samples on average for each utility value in the range.

6.4 Evolutionary Parameters

There are a number of parameters which need to be decided on when using an evolutionary algorithm. We largely settled on using "standard" values, based on past experience from previous projects. Other values were more a matter of practicality, based on the computational limitations and the limited time span available to us. The values chosen are summarised in Table 9.

Evolutionary Settings

Parameter	*Value*
Number of Experiments	1000
Number of Generations	100
Population Size	100
Initial Parents	20
Selection Method	Elitism
Mutation Rate	0.05
Co-evolution Evalutations	5
Stigmergy Updates	10

Table 9: Parameters of the artificial evolution algorithm.

We chose to run a thousand trials for each experiment, enough that it would take several hours to complete, but not so much as to tax our capacities. We would typically be running about 3 experiments per night, and another couple while we worked on other computers.

We settled on a conventional 100 generations per trial. From experience we expected the fitness increase from additional generations to start levelling off after about 50 generations, so that more than a hundred generations would generally be a waste of time. In order to be sure of this, however, we chose to run one experiment for each scenario with 200 generations.

The population size was fixed at one hundred, which is usually a decent number in our experience. Maintaining a diverse population is necessary for a successful evolutionary algorithm, but a too large population can slow things down.

With generational mixing, as we had chosen to implement for this project, the biggest determinant for how quickly the population converges is the ratio between the population size and the number of children generated each generation. Given the nature of our crossover process (described in section 4.5.4), the number of children generated would vary considerably, so we needed to attempt crossover relatively often. As such we settled on making 20 crossover attempts per generation.

Our choice of selection method is detailed in section 5.6.1. We tried a number of different options before settling on the simple Elitism selector, which would keep the best 100 genomes after each generation.

We picked a relatively low mutation rate, considering we got the highest utility when we had a mutation rate of around 50%. As seen in section 5.6.2, once we had solved the major issues with

excessive cloning, there was not much difference in utility unless the mutation rate was very low or very high. We settled on 5%, reasoning that it would mean on average one mutant per generation, with a potential for many more if it happened upon a mutation with a lot of crossover potential.

6.4.1 Full Communication

In order for our Full Communications evolver to determine the utility of a genome, it would need to evaluate how well it worked together with genomes from the other BEEs. Each genome would have to be evaluated multiple times, to ensure that the fitness it received was not dominated by bad luck from being paired with poor options.

But on the other hand, these simulations would take up a significant amount of time, and would likely be the dominating factor in how long the experiment took, so we needed to limit the number of evaluations as much as possible. We decided to run at least five evaluations of each genome. We would, however, also run an experiment with ten evaluations, in order to determine whether five was enough or not.

In total, then, we would run three Full Communication experiments:

- The first would be a standard experiment, using the parameters as listed in Table 9.
- The second would test the potential improvements from more fitness evaluations per genome, and so would have a minimum of 10 evaluations per genome instead of 5. Otherwise remaining as described in the table.
- The third would test whether the we were terminating the evolutionary process to soon, and would have 200 generations, instead of the original 100.

6.4.2 Stigmergy

The stigmergy approach developed by Tripp and Palmer[7] is based on the swarm occasionally receiving updates on each others actions and plans. The frequency of such updates determines how reactive the swarm is, and also how often they re-plan. The re-planing process takes a lot of computational power, with each stigmergy update being equivalent to an entire No Communications trial.

Given that the average length of a transfer within the asteroid belt is about 2 years, we eventually settled on 10 stigmergy updates, which would come 2 years apart in simulation time. Anything more than that, and there would not be any changes to report in-between updates, which would simply waste computing power. So we picked the highest practical number, in order to not limit Stigmergy more than necessary. This level of communication would require one broadcast from Earth every two years, so would still be a significant communication saving over constantly remaining in contact.

Stigmergy Behaviour Settings

Greedy	*Considerate*	*Proactive*	*Obstinate*
1	1	1	1
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	0	0	0
0.5	0	1	1
0.5	1	0	1
0.5	1	1	0

Table 10: The behavioural parameters of the BEEs when running the Stigmergy method.

The fitness function of each BEE is based on four behaviours, as described in section 3.5: Greedy, Considerate, Proactive, and Obstinate. In order to function at the highest efficiency, different BEEs

should assign different weights to each behaviour. In situations with overlap, that should help them adjust to avoid it without losing any efficiency. In order to test whether the Stigmergy behaviour is actually helping the result, rather than the extra generations of evolution, we also run a homogeneous variant. In this version, the BEEs only listen to the Greedy component of their fitness function, so that they do not take the new information about the environment into account.

For the standard heterogeneous experiment, we assigned each BEE different behaviours based on different binary possibilities, as seen in Table 10. The current thinking was that the Greedy behaviour would be necessary to some extent, which is why the last three BEEs break the pattern; being assigned 0.5 instead of the 0 which would have been expected. We had hoped to have time to tweak these behaviours later, but never found the time. The Greedy variant used for the second experiment assigned a weight of 1.0 to Greedy and zero to the rest for each BEE.

Like with the Full Communications evolver, we would run three experiments:

- The first, standard, experiment would use the distribution of behaviours from Table 10, and the base evolutionary parameters from Table 9. This would be the main test of the stigmergy approach.
- The second experiment would be the homogeneous all Greedy variant. Like the first experiment, it would use the evolutionary parameters from Table 9. It would provide a data point to differentiate between Stigmergy's effect on the scientific return and the effect of the simple ability to improve plans as they went along.
- The third experiment would be a 200 generation experiment to test whether we were terminating the evolution too soon. Since there was no difference in computational complexity in the first two experiments, it would be run with whichever behaviours functioned best.

6.4.3 No Communication

Our last level of communication was designed to test the efficiency cost from having no communication at all. The plan was to use heterogeneity in the fitness evaluations of the individual BEEs to make it less likely for them to pick the same options. If they naively tried to maximise their own contribution to the mission's scientific return, they should end up taking the same options, and thus producing overlap. Naturally, we would have to test that assumption to be sure of whether deliberately picking sub-optimal solutions actually produced a better run or not.

The heterogeneous weights were copied from the ones used by stigmergy; again with the intention of improving them at a later time. While the behaviours are different, they still needed to be diverse and there happens to be the same number of them, so the same array of weighs could be reused.

The experiments are similar to the ones done for Stigmergy above:

- The first experiment would use the standard fitness behaviours from Table 11, in addition to the normal evolutionary parameters from Table 9. It would test the no communication approach we had come up with.

No Communication Behaviour Settings

Greedy	Delta V	*Time*	*Varied*
1	1	1	1
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	0	0	0
0.5	0	1	1
0.5	1	0	1
0.5	1	1	0

Table 11: The behavioural parameters of the BEEs when running the No Communication experiment.

- The second experiment was designed to test the necessity of heterogeneous agents. The agents would be fully Greedy, caring only to maximise their contribution to the scientific return of the mission. It would otherwise be the same as the first experiment.
- The third experiment was, as with Full Communication and Stigmergy, designed to test for premature termination. Since there again was no difference in computational complexity in the first two experiments, it would run 200 generations with whichever behaviours functioned best.

6.5 Route Lengths and Overlap

In order to test whether overlap actually is prevalent, we will compare the amount in our various experiments. Specifically, the simulator will be modified to count how many times a spacecraft visits an asteroid which has already been explored. Due to computational limitations, we only plan to do this for our three standard experiments, the two purely greedy experiments and the Monte Carlo baseline.

The number of unique routes in the Transfer Map will also be counted and sorted according to length. This will provide some vital context about the nature of the solution space itself and an indication as to whether the evolutionary algorithm or Monte Carlo is finding the best options.

7 Results

This chapter presents our results in a variety of tables and graphs. The first four sections below provide the results for each of our communication levels and baseline in turn. Section 7.5, meanwhile, collects results from different communication levels into comparative graphs. Each of the first four sections are organised as follows:

The first table in each section provides a basic overview of the performance of each experiment. The averages and standard deviations for both scientific return and number of asteroids visited are listed. The table also includes how many trials were run for each experiment.

The second table provides more detailed statistics about the distribution of results. The quartiles, min and max values, and the interquartile range are presented for both scientific return and asteroids visited. Figures 31 and 32 on page 68 aggregate these statistics in a box and whiskers plot for easier comparison.

Next follows a histogram for the scientific return of each experiment. The bin size for these graphs is 1 point of scientific return, so that each column represents the number of result which would be rounded to that integer. A column graph of the number of unique asteroids visited by each experiment is also presented.

The final graph in each section is a scatter plot depicting every trial we have run for that level of communication. In order to improve readability, the number of asteroids visited for each trial is modified by a random variable (up to +/-0.5). As the number of asteroids is always an integer value, they would otherwise form difficult to read lines rather than clouds. The Monte Carlo graph is not included, as rending a million data-points proved too resource intensive.

7.1 Baseline: Monte Carlo Simulation

Monte Carlo Performance

Trials	1,000,000	
	Science	*Asteroids*
Average	384.66	30.61
Standard Deviation	23.79	2.01

Table 12: The average and standard deviation from a Monte Carlo Simulation of the solution space.

Monte Carlo Quartiles

	Science	*Asteroids*
Min	247.36	22
25th Quartile	369.22	29
Median	385.11	31
75th Quartile	400.72	32
Max	499.85	40
Interquartile Range	31.50	3

Table 13: The quartiles and related statistics from a Monte Carlo Simulation of the solution space.

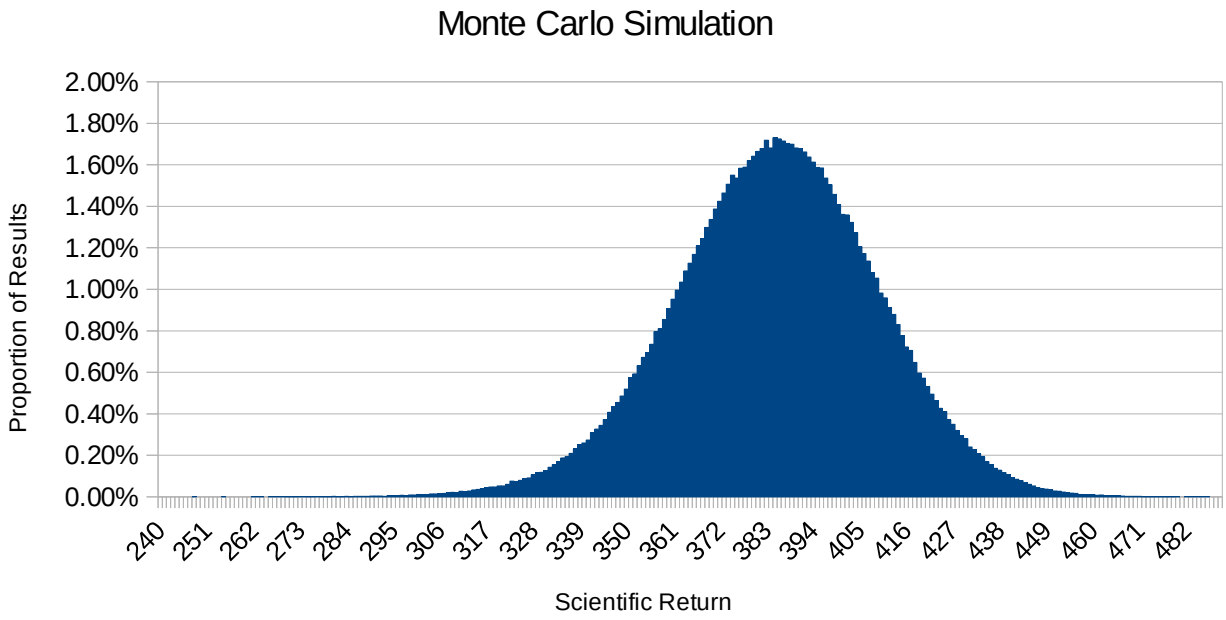

Figure 18: The distribution of scientific return from a Monte Carlo Simulation of the solution space.

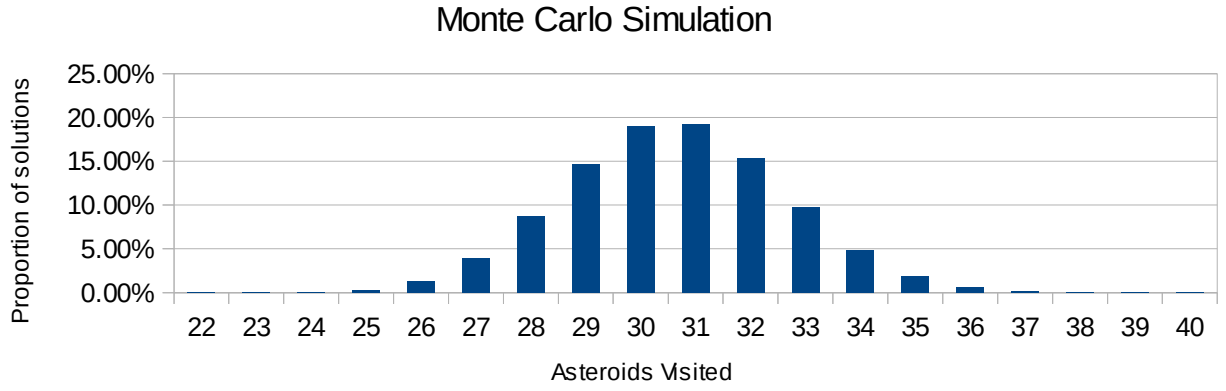

Figure 19: The distribution of asteroids visited from a Monte Carlo Simulation of the solution space.

7.2 Full Communication

Full Communication Performance

	Standard Experiment		*10 co-evolution evaluations*		*200 Generations*	
Trials	1,000		1,000		1,000	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Average	659.20	48.17	670.80	48.70	665.55	48.41
Standard Deviation	15.38	1.57	21.21	1.58	15.88	1.58

Table 14: The average and standard deviation from a basic Full Communication run, a run with twice the co-evolution evaluations, and a run with twice as many generations.

Full Communication Quartiles

	Standard Experiment		*10 co-evolution evaluations*		*200 Generations*	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Min	613.51	44	618.44	43	612.23	43
25th Quartile	648.85	47	660.09	48	654.41	47
Median	659.08	48	669.88	49	664.46	48
75th Quartile	669.41	49	681.30	50	676.16	49
Max	709.70	54	715.60	53	711.50	54
Interquartile Range	20.56	2	21.21	2	21.76	2

Table 15: The quartiles and related statistics from a basic Full Communication run, a run with twice the co-evolution evaluations, and a run with twice as many generations.

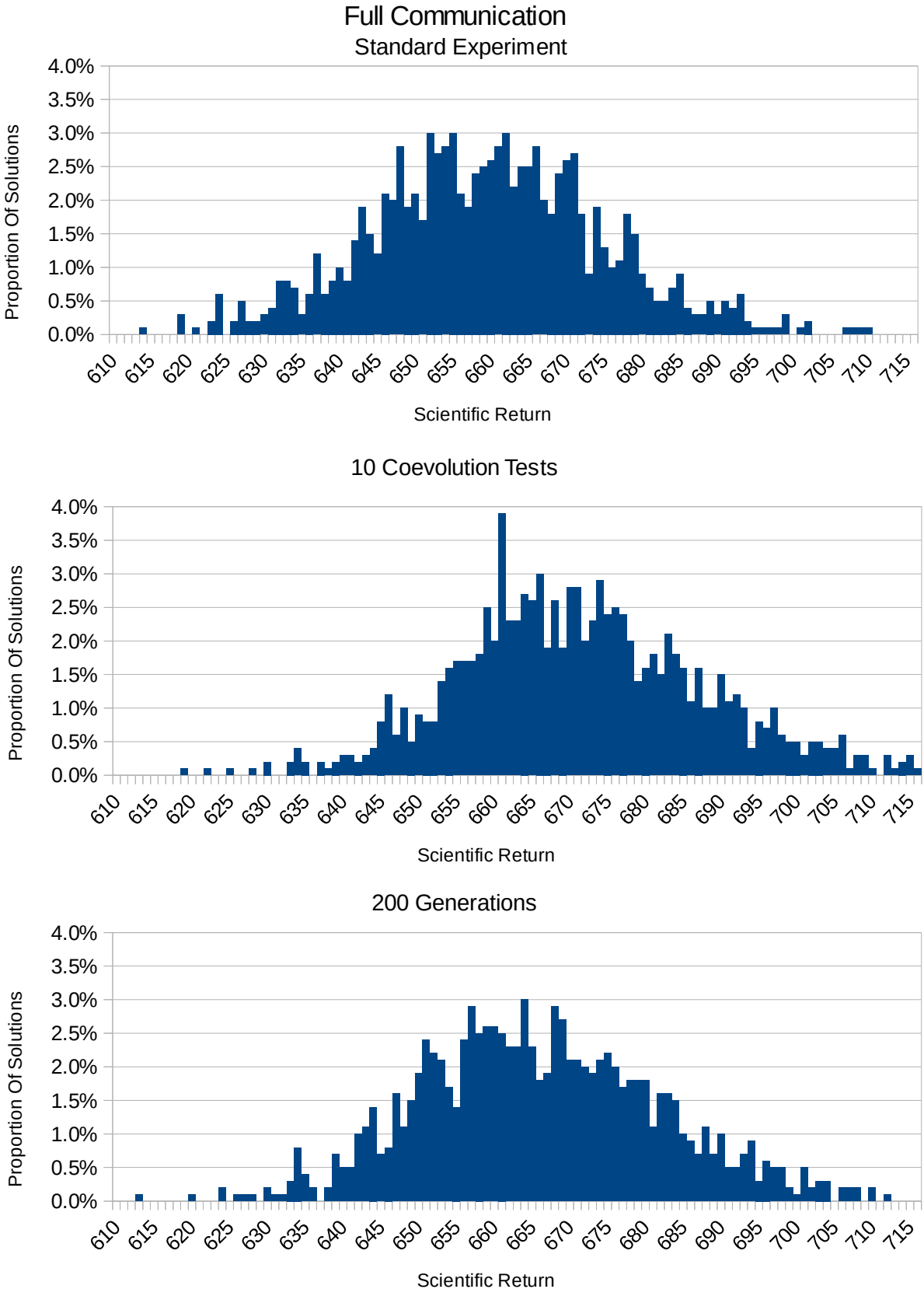

Figure 20: Distribution of scientific returns from a basic Full Communication run, a run with twice the co-evolution evaluations, and a run with twice as many generations.

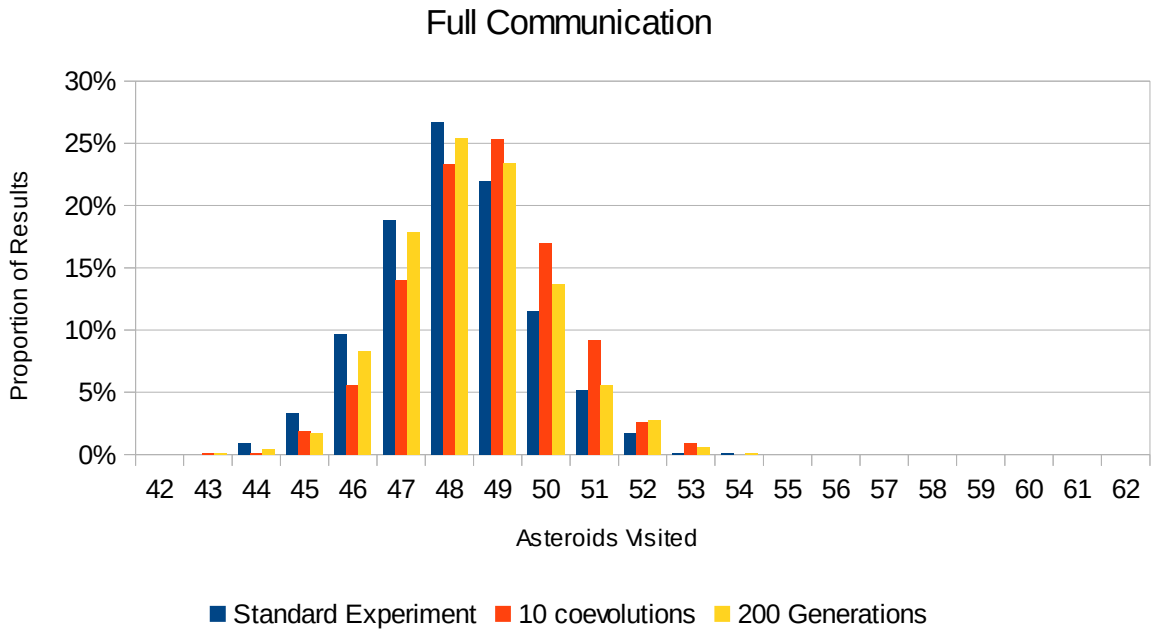

Figure 21: Distribution of the number of asteroids visited from a basic Full Communication run, a run with twice the co-evolution evaluations, and a run with twice as many generations.

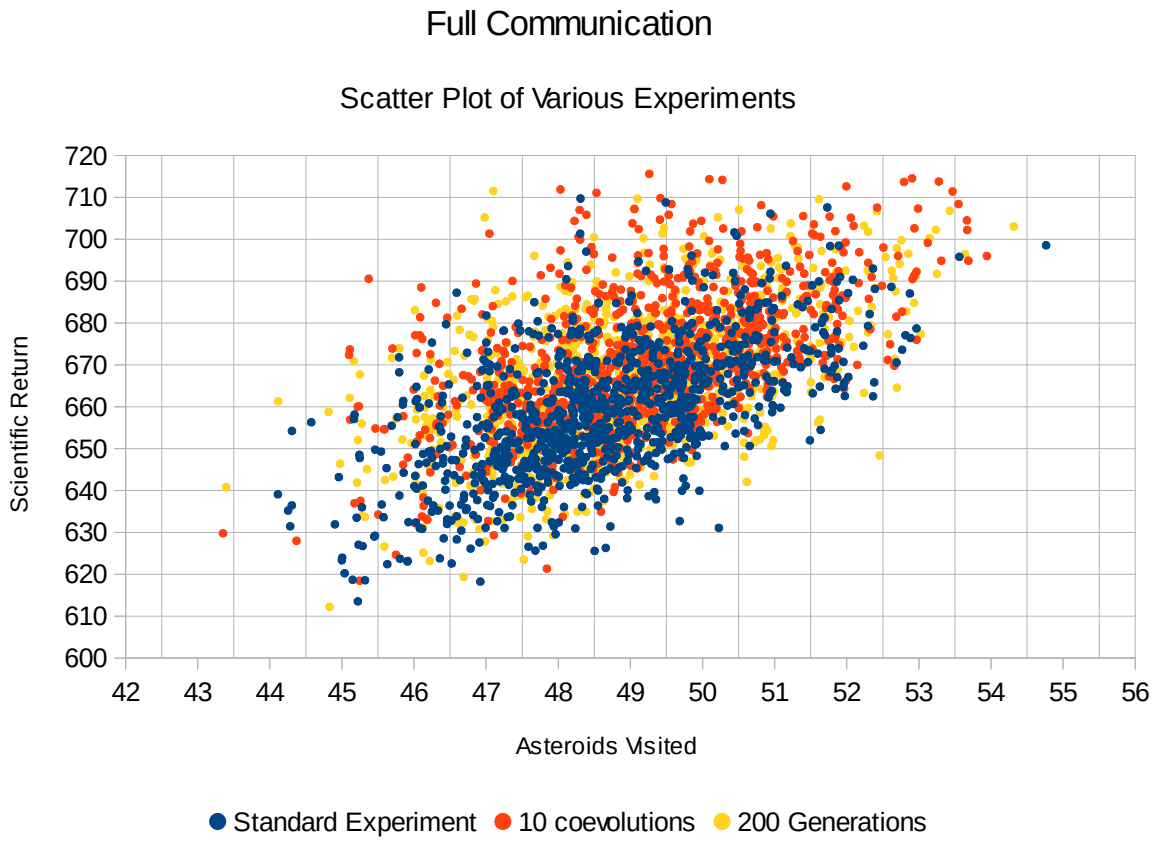

Figure 22: Scatter plot of our results, from a thousand trials of each: a standard Full Communication experiment with 5 co-evolution evaluations and 100 generations, a variant with twice the examinations of each genome and a version with twice the generations.

7.3 Stigmergy

Stigmergy Performance

	Standard Experiment		*Homogeneous Greedy Agents*		*200 Generations, Greedy Agents*	
Trials	1,000		1,000		1,000	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Average	664.07	54.36	686.18	54.57	694.16	55.68
Standard Deviation	19.01	2.05	18.58	2.18	19.72	2.09

Table 16: The average and standard deviation from a basic Stigmergy run, a run with all greedy agents, and a 200 generation run with all greedy agents.

Stigmergy Quartiles

	Standard Experiment		*Homogeneous Greedy Agents*		*200 Generations, Greedy Agents*	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Min	583.52	48	617.29	46	625.90	48
25th Quartile	651.79	53	674.68	53	681.46	54
Median	665.05	54	686.90	55	694.63	56
75th Quartile	676.83	56	699.56	56	707.83	57
Max	722.97	61	738.21	60	749.75	61
Interquartile Range	25.04	3	24.88	3	26.37	3

Table 17: The quartiles and related statistics from a basic Stigmergy run, a run with all greedy agents, and a 200 generation run with all greedy agents.

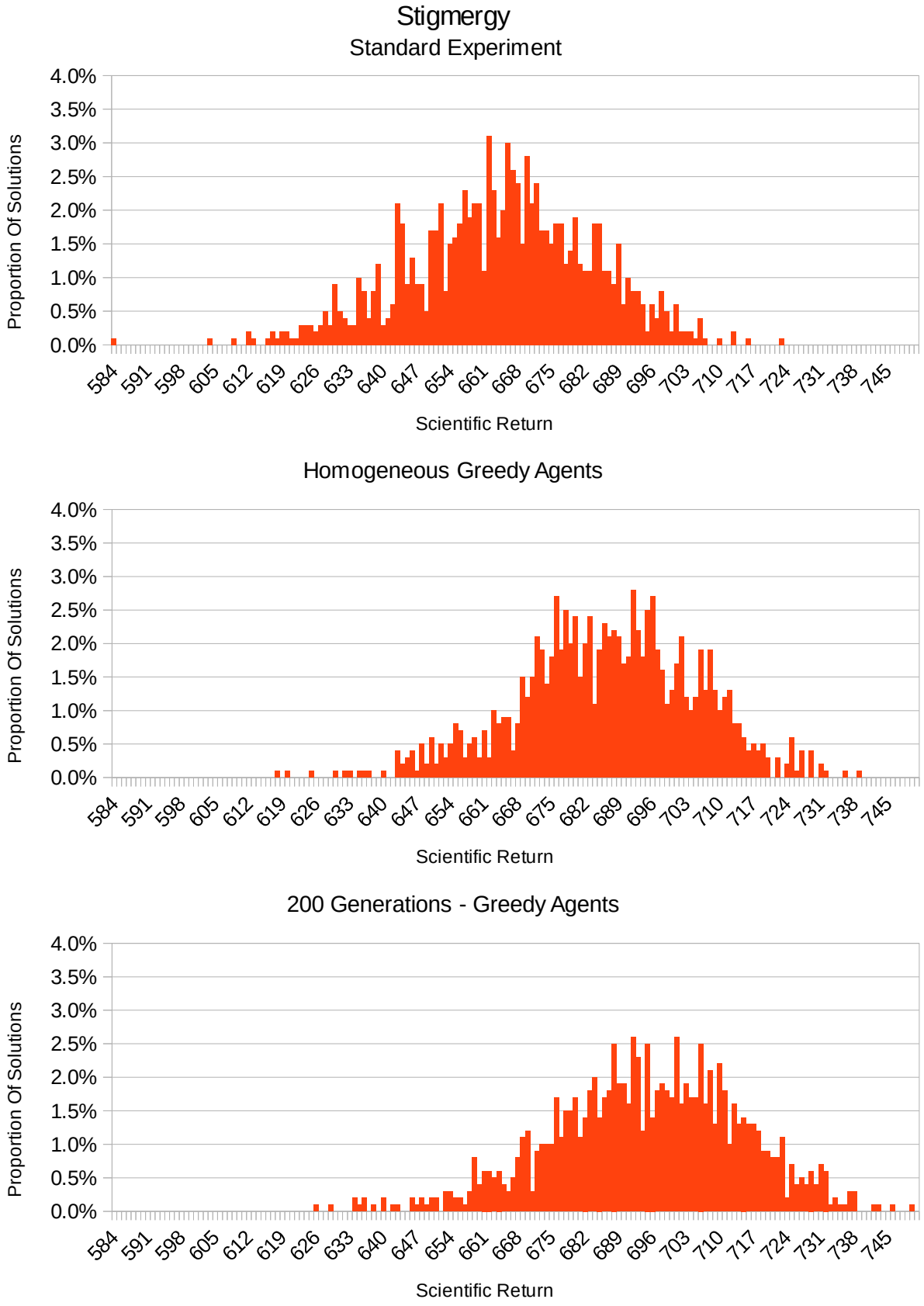

Figure 23: Distribution of scientific returns from a basic Stigmergy run, a run with all greedy agents, and a 200 generation run with all greedy agents.

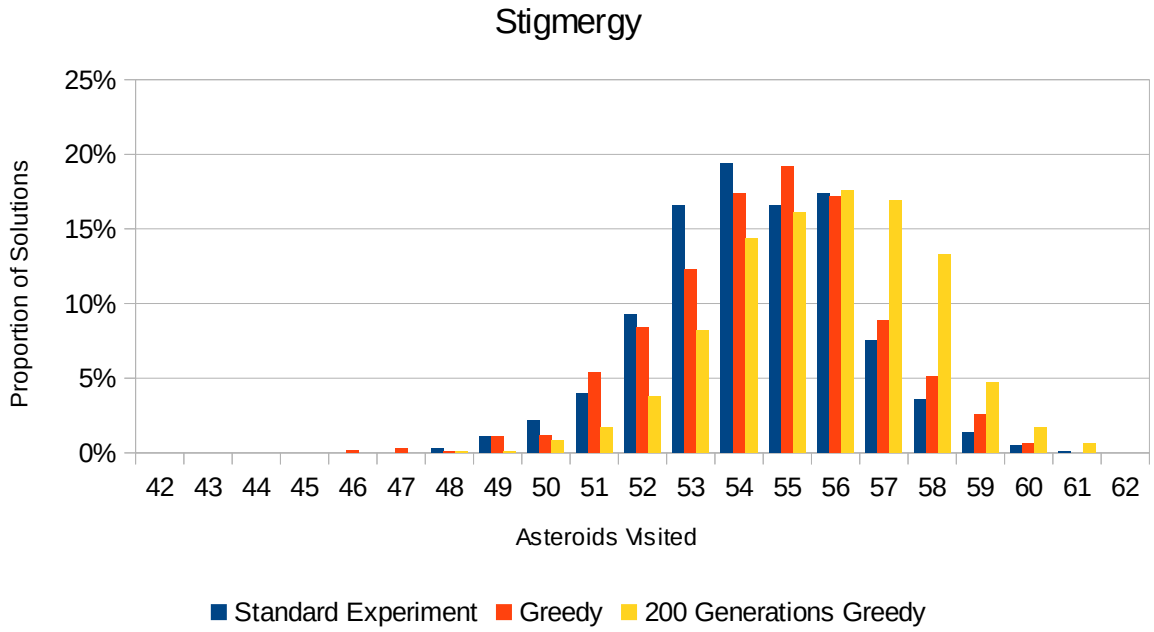

Figure 24: Distribution of the number of asteroids visited from a basic Stigmergy run, a run with all greedy agents, and a 200 generation run with all greedy agents.

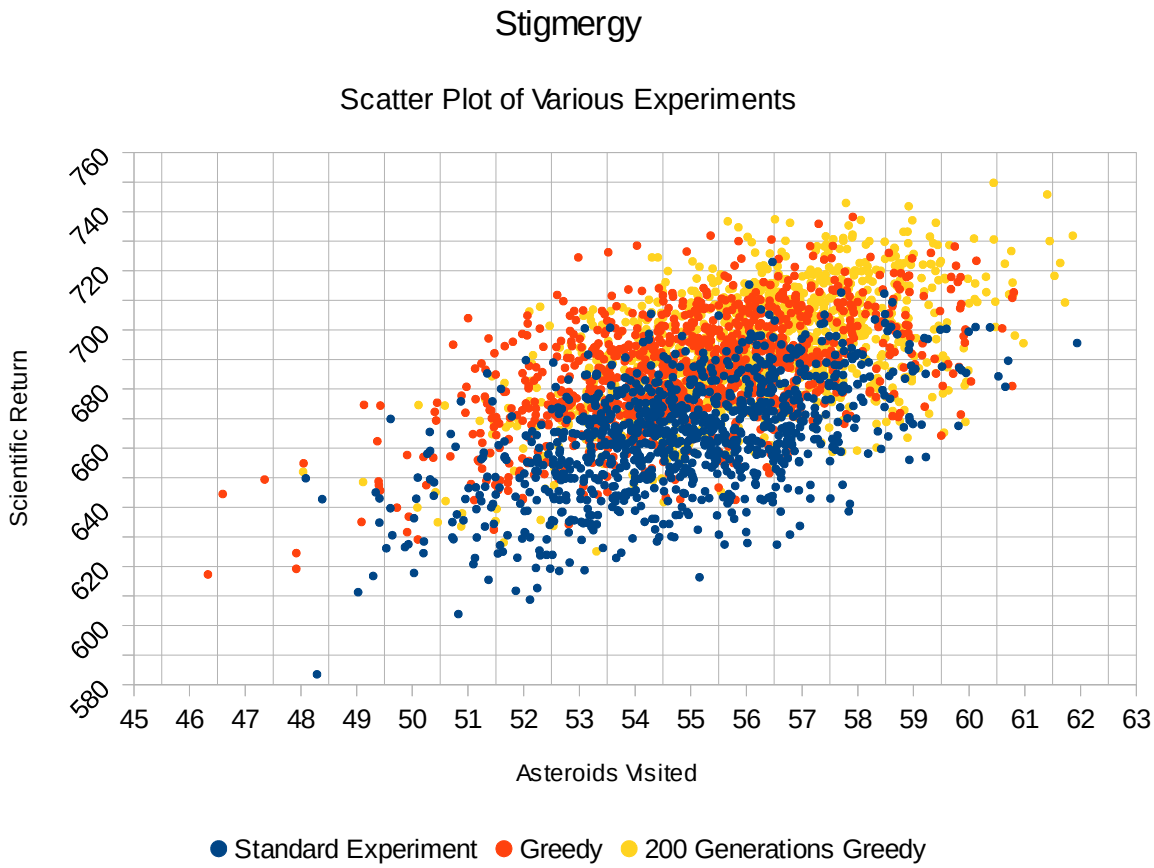

Figure 25: Scatter plot of our results, from a thousand trials of each: standard heterogeneous experiment, a homogeneous (all greedy) experiment, and a 200 generation experiment with all greedy agents.

7.4 No Communication

No Communication Performance

	Standard Experiment		*Homogeneous Greedy Agents*		*200 Generations, Greedy Agents*	
Trials	1,000		1,000		1,000	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Average	665.38	54.40	687.04	54.57	694.52	55.78
Standard Deviation	13.81	1.97	18.24	2.16	19.51	2.12

Table 18: The average and standard deviation from a basic No Communication run, a run with all greedy agents, and a 200 generation run with all greedy agents.

No Communication Quartiles

	Standard Experiment		*Homogeneous Greedy Agents*		*200 Generations, Greedy Agents*	
	Science	*Asteroids*	*Science*	*Asteroids*	*Science*	*Asteroids*
Min	587.04	47	614.00	46	625.59	49
25th Quartile	653.18	53	675.37	53	681.55	54
Median	665.86	54	687.94	55	694.74	56
75th Quartile	678.26	56	699.42	56	707.30	57
Max	720.28	60	738.41	61	751.92	62
Interquartile Range	25.08	3	24.05	3	25.76	3

Table 19: The quartiles and related statistics from a basic No Communication run, a run with all greedy agents, and a 200 generation run with all greedy agents.

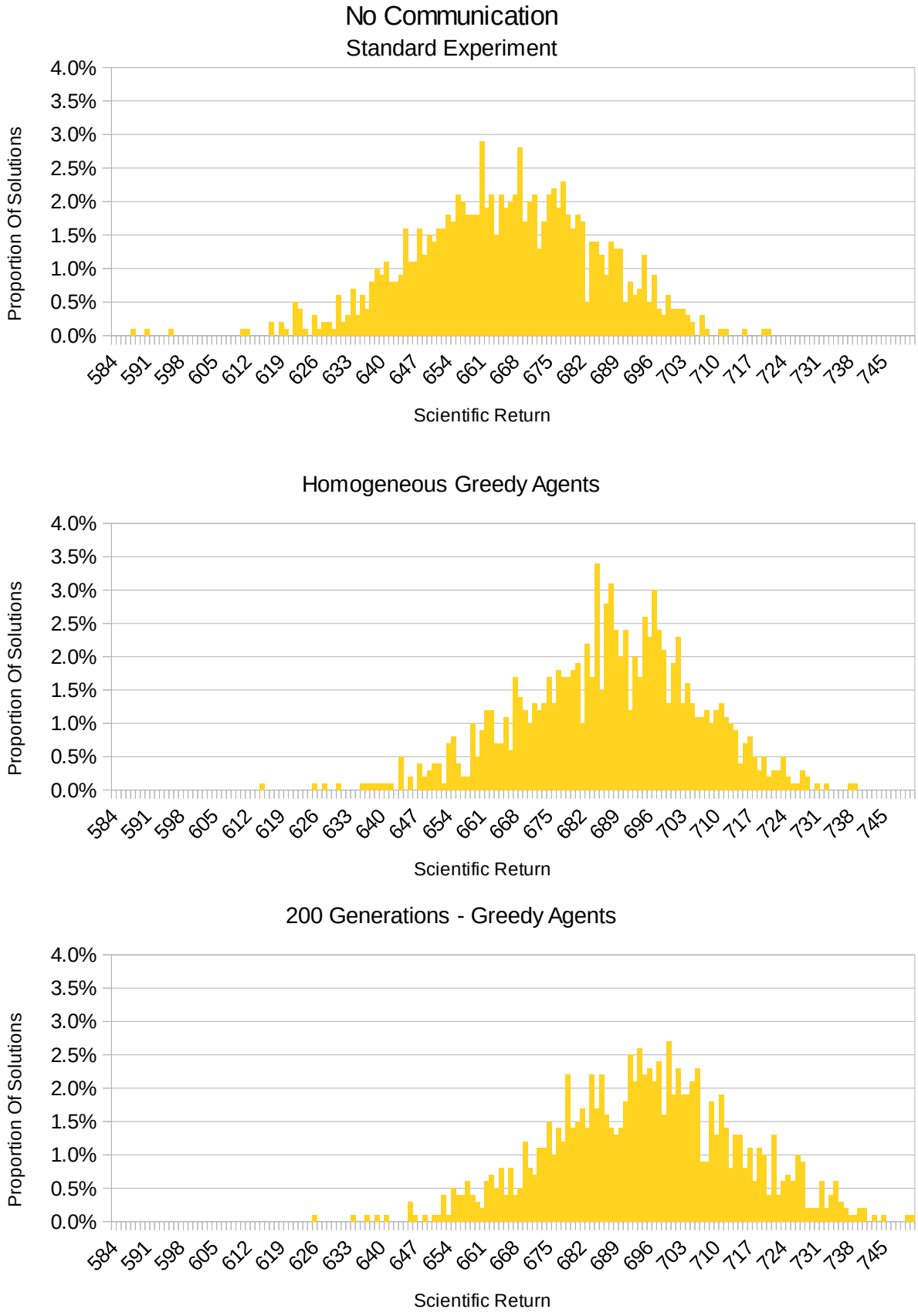

Figure 26: Distribution of scientific returns from a basic No Communication run, a run with all greedy agents, and a 200 generation run with all greedy agents.

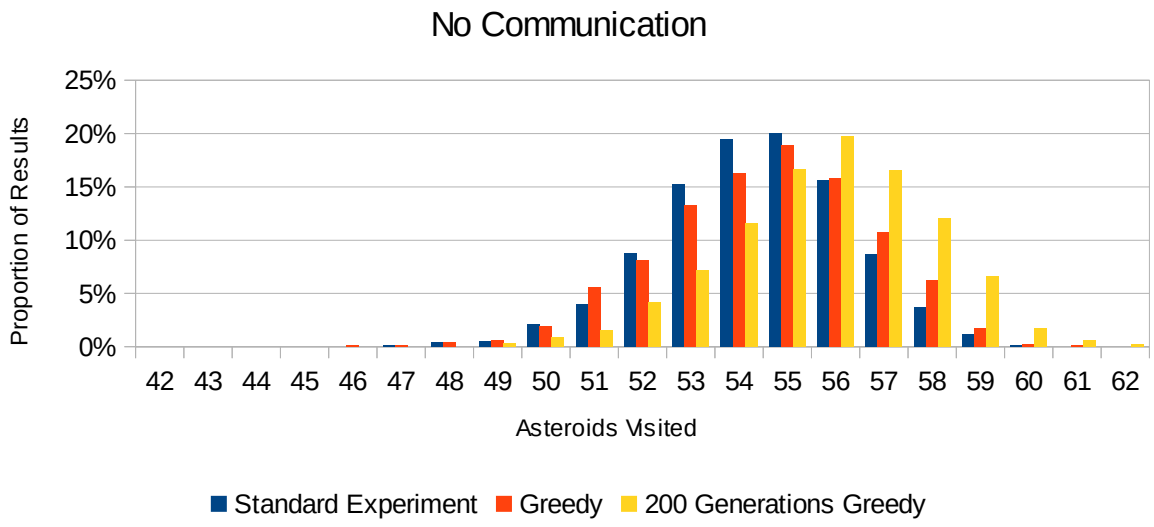

Figure 27: Distribution of the number of asteroids visited from a basic No Communication run, a run with all greedy agents, and a 200 generation run with all greedy agents.

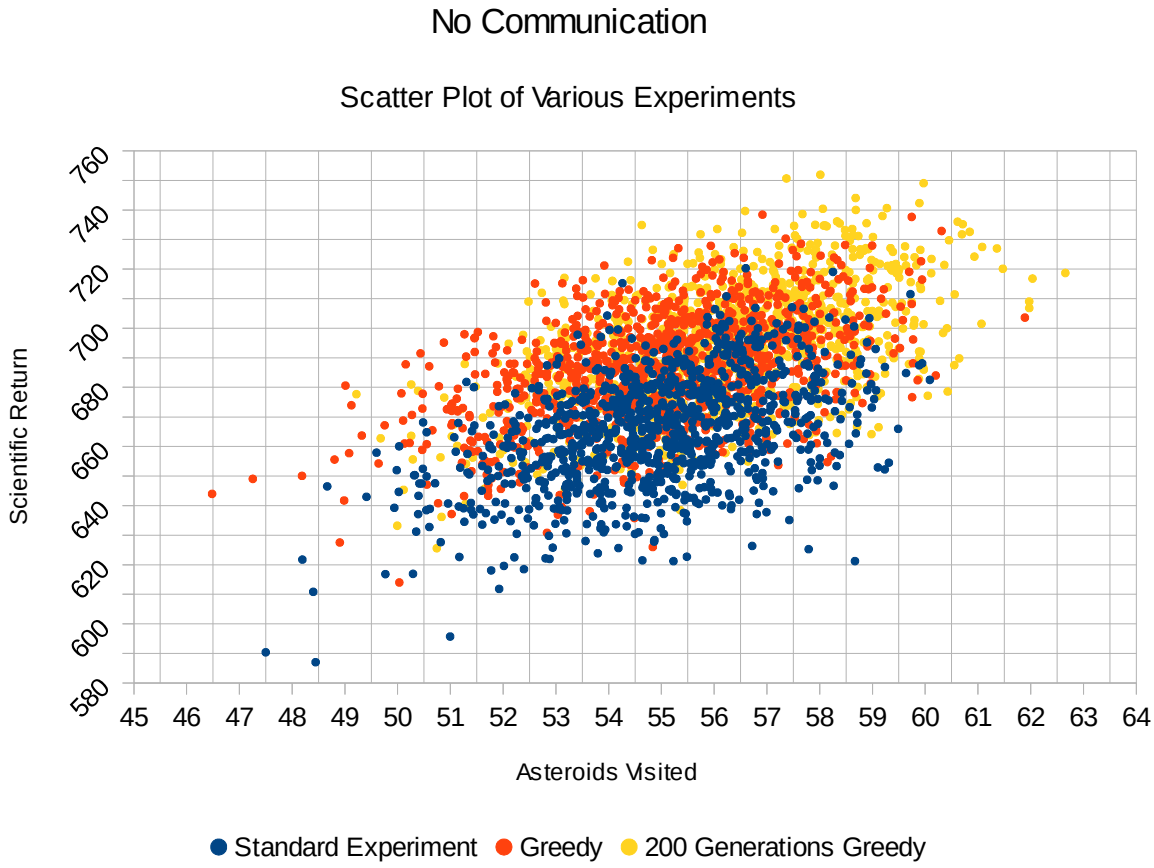

Figure 28: Scatter plot of our results, from a thousand trials of each: standard heterogeneous experiment, a homogeneous (all greedy) experiment, and a 200 generation experiment with all greedy agents.

7.5 Combined Data

This section provides a bit more context for the data presented so far. Here we have included experiments from multiple communication levels in the same graphs, so that they can be more easily compared and contrasted.

Figure 29 shows a scatter plot of our best experiment from each communication level. This is defined as the experiment which produced the highest average scientific return, though the experiments in question also have the highest mean and the highest maximum. As with the scatter plots for individual communication levels, the asteroids visited have been modified by a random variable(up to +/-0.5). This was done in order to improve readability.

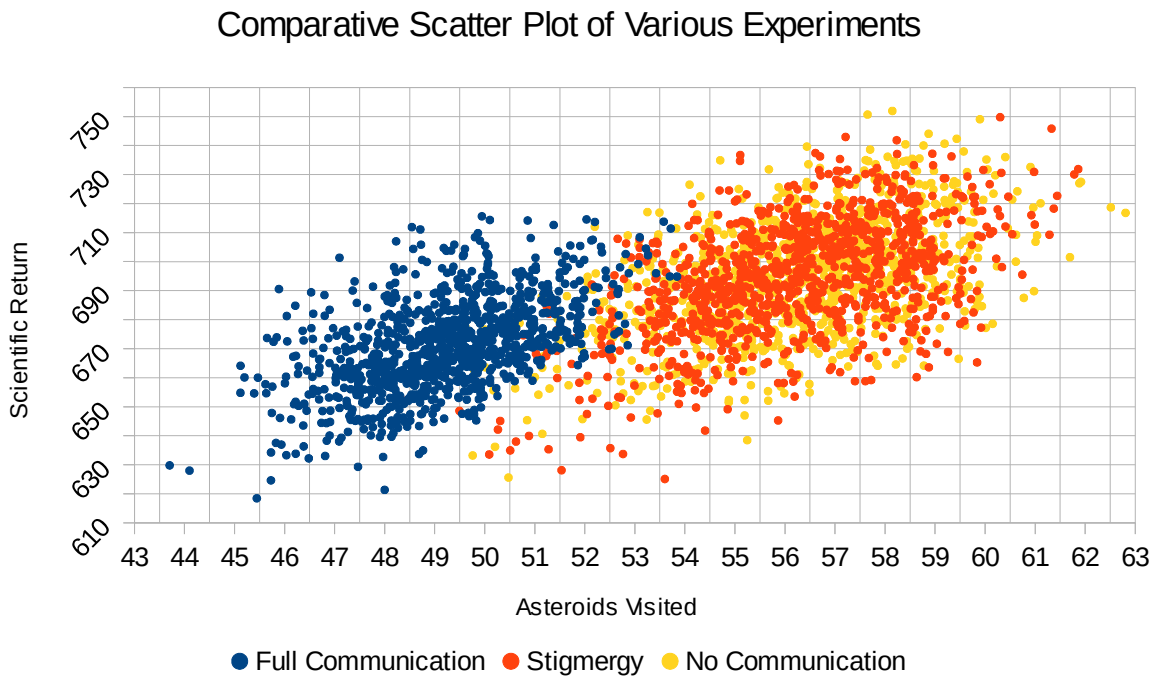

Figure 29: Scatter plot of our best experiments within each communications level, from a thousand trials of each: Full Communication with a hundred generations and at least 10 evaluations of each genome, Stigmergy with homogeneous greedy agents and 200 generations, and No communication with homogeneous greedy agents and 200 generations.

Figure 30 compares the distribution of scientific return from the same three experiments. In this histogram, the bins cover a ten point range, so that the 660 bin contains all results from 660-670 (exclusive). The communication levels are presented in the same colours as they are in the rest of the chapter.

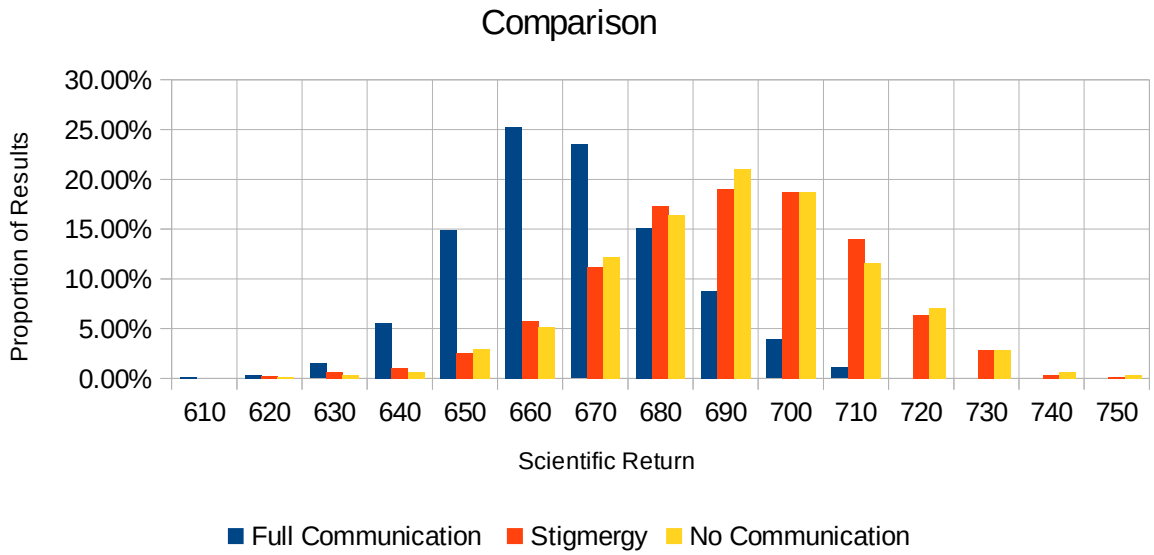

Figure 30: Histogram of the scientific return of our best experiments within each communications level: Full Communication with a hundred generations and at least 10 evaluations of each genome, Stigmergy with homogeneous greedy agents and 200 generations, and No communication with homogeneous greedy agents and 200 generations.

Figures 31 and 32 present the quartile statistics from above in a more readily compared fashion. In each experiment, the first line represents the first quartile (from the minimum to the 25th quartile). The box represents the inter-quartile range, within which the middle 50% of results were found. The line inside the box is the median result, and the last line is drawn between the 75th quartile and the maximum value.

Figure 31: Box and Whiskers plot comparing all the experiments run in terms of scientific return.

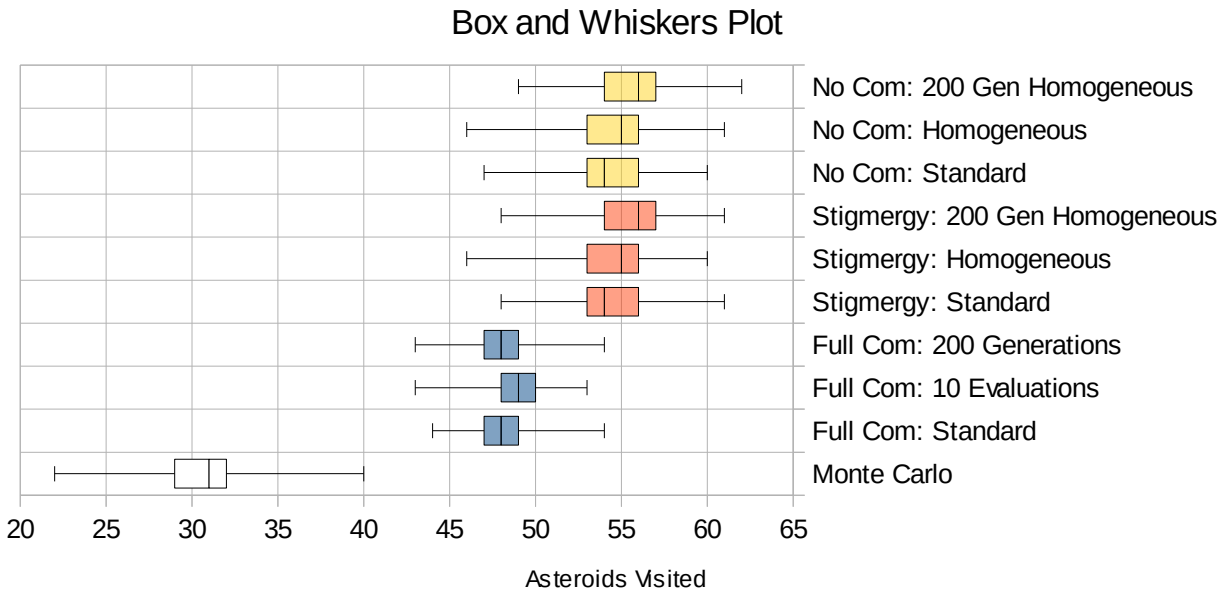

Figure 32: Box and Whiskers plot comparing all the experiments run in terms of asteroids visited.

7.6 Overlap

Table 20 contains the statistical information about overlap in our experiments. It provides the average and standard deviation for each experiment we tested overlap for and the quartile statistics for the same experiments. The distribution of overlap is graphed in Figure 33.

Overlap Statistics

	Monte Carlo	*Full Com: Standard*	*Stigmergy: Standard*	*Stigmergy: Greedy*	*No Com: Standard*	*No Com: Greedy*
Min	0	0	0	0	0	0
25th Quartile	0	0	1	1	0	1
Median	0	0	1	1	1	1
75th Quartile	0	0	2	2	2	2
Max	4	2	9	9	8	10
Interquartile Range	0	0	1	1	2	1
Average	0.08	0.06	1.6	1.5	1.4	1.6
Standard Deviation	0.29	0.24	1.4	1.5	1.3	1.4

Table 20: The overlap statistics and quartiles from our least computationally intensive experiments. Each point of overlap represents a BEE visiting a single already explored location.

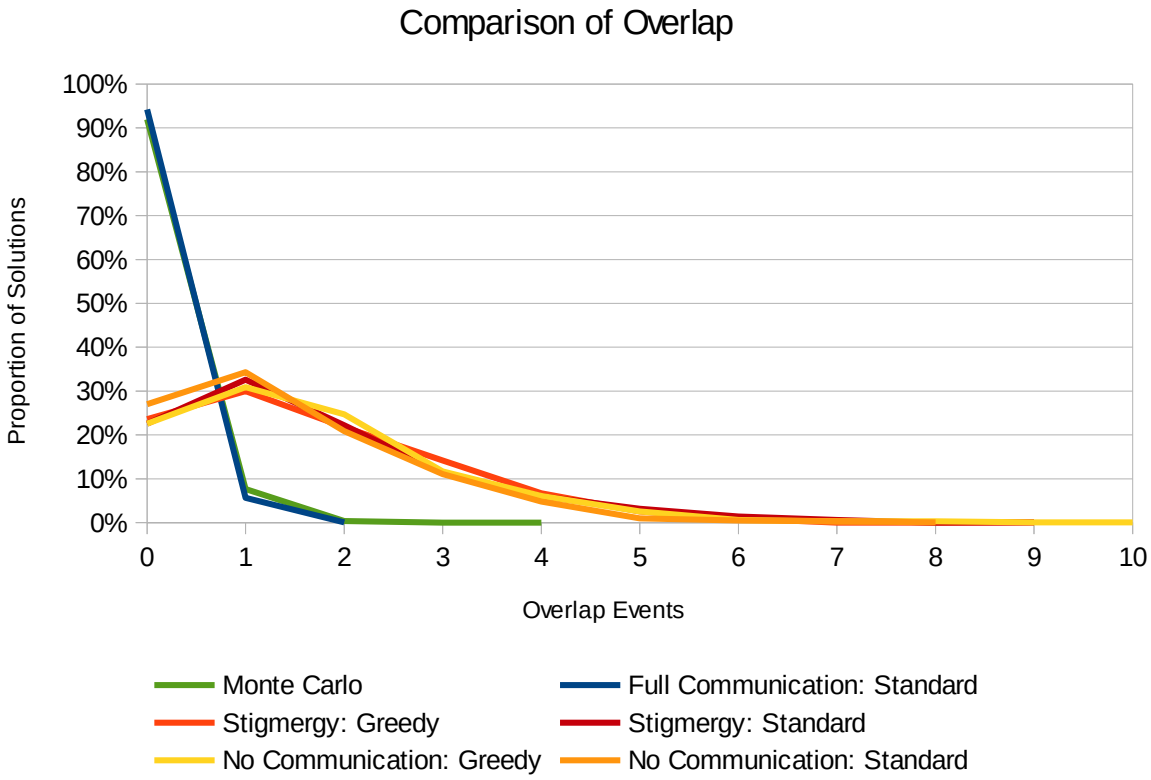

Figure 33: A comparison of the distribution of overlap levels in our six least computationally intensive experiments.

7.7 Route Lengths

Table 21 shows the routes and the beginnings of routes in the solution space. The table shows how many unique asteroids can be reached by routes up to a given length, how many complete (non-expandable) routes there are of a given lengths, and how many routes there are of that length which still has available transfers away from it. Note that we have pruned out all length one routes, as described in section 5.6.2.

Route Length Statistics

Route Length	*Reachable Asteroids*	*Number of Complete Routes*	*Number of Incomplete Routes*
1	2,446	0	3,471
2	33,547	901,752	2,900,543
3	42,128	925,755,810	287,715,499
4	42,128	10,540,976,077	404,447,454
5	42,128	3,459,381,073	31,149,669
6	42,128	108,000,354	278,288
7	42,128	500,479	318
8	42,128	470	0
Sum	42,128	15,035,516,015	726,495,242

Table 21: A count of how many unique asteroids can be reached from the HIVE by routes of at most a given length, the number of complete routes which have a given length, and the number of route segments of a given length which can be expanded into longer routes.

8 Discussion

The solution space had to be altered from the one originally envisioned due to the computational problems that plagued the Transfer Map Generator (see Appendix B). The routes least likely to have good utilities were excluded from the solution space with the result that the expected utility of a random route is higher for our solution space. We were also unable to completely map the reduced solution space, requiring the experiments to be run on an even smaller subset.

The implementation can be summarised as the process of discovering how severely we underestimated the problem size and attempting to handle the unrealistically large run time needed to generate the map. The difficulty was not finding a solution, it was finding a good and efficient solution. Most of the development time was spent reducing the problem size and finding techniques for improving our algorithms by exploiting properties of the problem and solution space.

The first issue that cropped up was the implementation of the search algorithm, which was quickly found lacking in terms of performance. The problem was not so much the time needed to compute the transfer windows for a single asteroid pair; a few days of optimisations had reduced it down from 15 minutes to roughly a millisecond. The problem was rather the large number of asteroid pairs for which transfer windows would have to be computed; likely tens of billions. With these numbers, it would take many weeks, if not months to generate the map.

The full database of asteroids was filtered down from roughly 650,000 asteroids to roughly 210,000. Asteroids were first ordered according to semi-major axis and removed in ascending and descending order until one was found that could be reached from the HIVE. The same process was repeated for inclination, removing the asteroids with the highest difference in inclination. The filtering reduced the problem size by roughly an order of magnitude, but may have eliminated parts of the solution space. If so, the eliminated routes would be the ones which were the least likely to be chosen by the Evolutionary Algorithms, but equally likely to be chosen by the Monte Carlo simulation.

8.1 Solution Quality

In our hypotheses (section 1.3.1), we used a metric we called *solution quality* to examine how well our evolutionary algorithms would do compared with Monte Carlo and each other. The solution quality grades our results so that the worst result gets a score of zero percent quality, the best gets one hundred percent quality, and the remainder a score depending on how close they are to each of those. A solution which is 20% of the way between the lowest and the highest score, for example, would have a 20% solution quality.

Our results range from a scientific return of 247.36 to 751.92. These are roughly 500 points apart, so each percentage of solution quality would correspond with about 5 additional scientific return over the worst solution. Our 20% solution from above would thus have a scientific return of about 350 points.

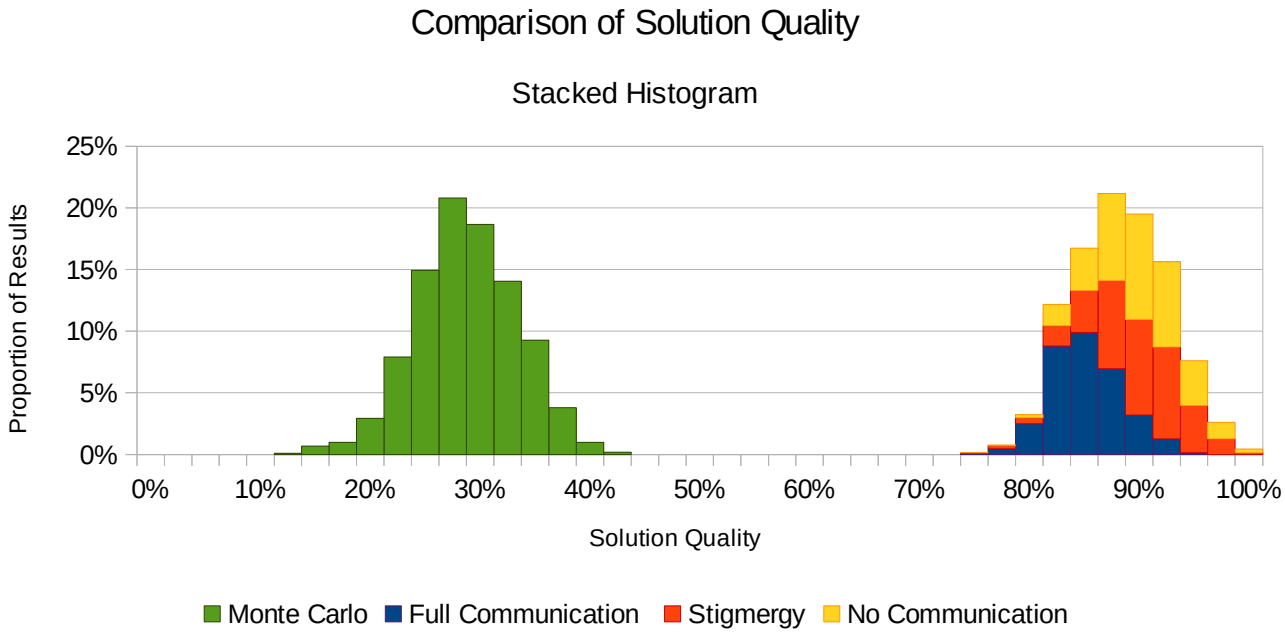

Figure 34: A comparison of the Monte Carlo and our AI results, in terms of solution quality as used in our hypotheses. The AIs have been scaled down so that the total volume of the histogram is the same as the Monte Carlo's volume.

Our results, in terms of solution quality, are summarised in Figure 34. As can be seen, the Monte Carlo simulation's results are centred at roughly 30% and ranges about 10% from that in either direction. The evolutionary algorithms are similarly spread around 90% ranging from about 80% to 100%.

8.2 General

The results did not show an improvement when using heterogeneous agents, quite the opposite. Figures 31 and 32 show that homogeneous greedy agents performed noticeably better than the heterogeneous ones. Regardless of the scenario, heterogeneity does not seem to provide a net benefit. This result is most easily explained by there being virtually no overlap. As this is the case, there is little to no benefit to be found using heterogeneity.

The heterogeneity as used in the No Communication scenario makes some spacecraft choose lower utility routes on purpose, assuming that other spacecraft have chosen the higher utility choice. The difference in priorities should ideally decrease the utility less than the duplication of effort would have done. However, our standard experiment does not seem to be reducing overlap by any noticeable amounts, suggesting that the weights as they were are not having the desired effect.

Stigmergy uses heterogeneity in a slightly different manner. The additional information allows spacecraft to know for sure whether others have made or currently plan to make the better choice and not simply assume so. The result should be a lower utility penalty as they should only divert from their perceived best choice if needed, and a higher benefit as they should be able to preemptively avoid overlap. The results instead show an overall reduction in utility when using heterogeneous agents, again without any reduced overlap.

Based on the significant difference in both utility and overlap between Full Communication and the other experiments, it seems that avoiding overlap (as the Full Communication evolver does) comes with a harsh penalty to total utility. It seems it is better for the swarm to make a cheap transfer to an already explored asteroid, in order to pick a different, high utility, path from there. As the Full

Communication evolver is aware of the overlap as soon as it happens, it seems it is unable to climb past that dip in fitness to create long enough routes to overcome the temporary setback.

8.3 Monte Carlo

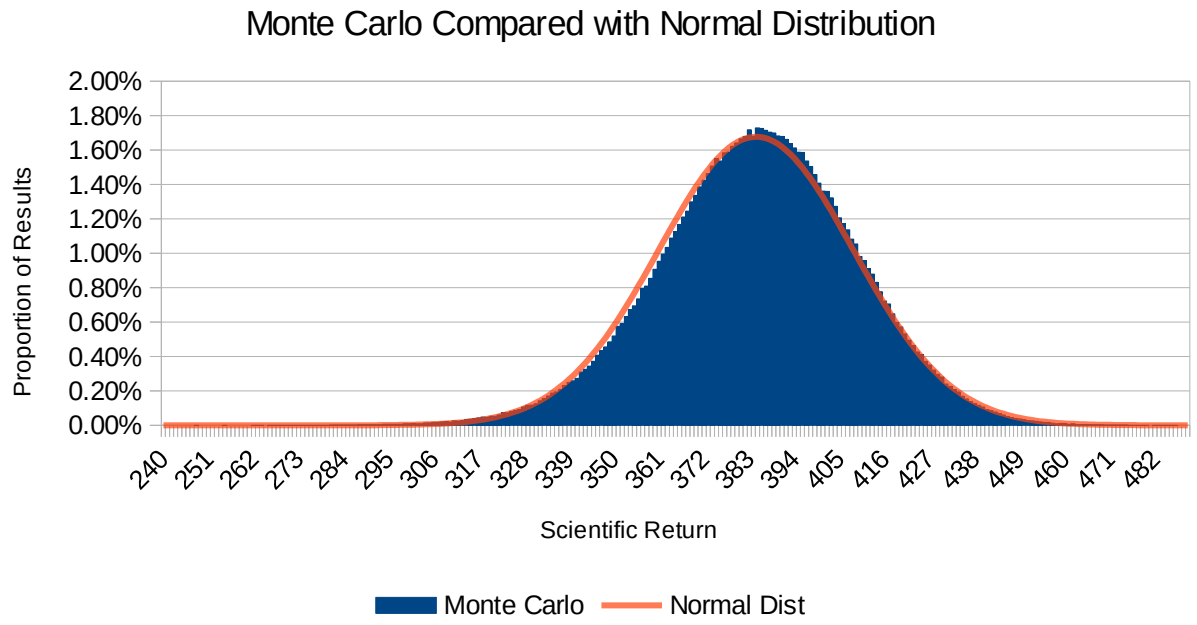

Figure 35: Monte Carlo results compared with a Normal Distribution with a mean of 384.66 (the Monte Carlo's average) and a standard deviation of 23.79 (same as the Monte Carlo).

In our hypothesis (section 1.3.1) we hypothesized that the Monte Carlo simulation would display an inverse relation between scientific return and proportion of results. Instead we found something which looked a lot closer to normally distributed, as can be seen in Figure 35.

There are a number of factors involved that account this discrepancy, illustrated in Figure 36. First of all, we made our hypotheses with the assumption that the BEEs would have the same amount of propellant (and thus delta v) as proposed in the APIES plan. As described in section 5.4, we have since over doubled the delta v available to the BEEs. Additionally, we assumed we would be searching for all possible transfers, rather than limiting ourselves to those beneath a given threshold cost (see section B.4).

The result is that our solution space is not, as we had originally thought, dominated by single transfer options where a single transfer would expend the entirety of the BEE's propellant reserve. Instead, routes with only a single asteroid have been pruned from the database (see section 5.6.2), and transfers were limited to 1,200m/s in delta v cost; just over a third of the total available.

As can be seen in Table 22, most randomly generated routes would visit 3 asteroids and over a 6th would visit even more. The result is that an average Monte Carlo run visits over 30 asteroids, where we had originally expected it to visit 19 or less. However,

Route Length Probability

Asteroids Visited	*Percent of Possible Routes*
1	0.00%
2	23.72%
3	58.20%
4	17.42%
5	0.66%
6	0.01%
7	$1.53 \cdot 10^{-5}$ %
8	$9.73 \cdot 10^{-9}$ %

Table 22: The odds of a randomly generated route being of a given length. Based on route length statistics from Table 21.

less than one percent of possible routes visit more than 4 asteroids. As such, the probability of all 12 BEEs randomly picking such long routes in the same Monte Carlo run is vanishingly small.

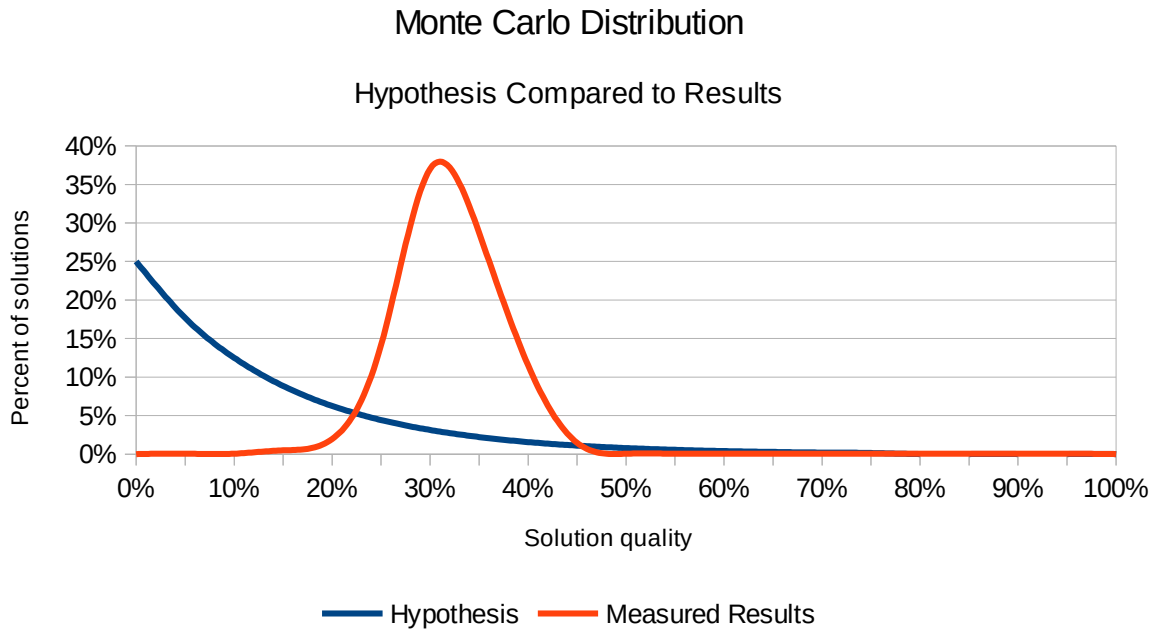

Figure 36: Our hypothesis (Figure 1) compared with the final result, in terms of solution quality.

It is further interesting to note that due to the transfer cost limits, there is a higher chance of generating a three asteroid route than a shorter two asteroid route. In our original hypotheses, we would have assumed that it would be strictly less probable to generate a longer route than a shorter one.

Monte Carlo, with a million trails, did not manage to find a single solution with a scientific return above 500. Of our nine thousand evolutionary trials, only four produced a scientific return below 600, and the lowest result (583.52) was significantly higher than the best produced by Monte Carlo. This tells us that high utility solutions are extremely rare in the data-set.

With a normal distribution it is possible to calculate the odds of finding a solution better than a given value. The chance of randomly locating a solution which has a scientific return of at least as good as the worst we found via evolution is 3.76E-17, or one in 26 quadrillion, assuming a mean and standard deviation as given in Figure 35. While it is unlikely that the actual mean and standard deviation happen to be exactly equal to those found from a million random samples, it is likely fairly close.

The extremely low chance of finding a good solution makes it largely infeasible to solve the problem using Monte Carlo. Even with an extremely powerful computer capable of completing one Monte Carlo run every nanosecond, it would take roughly a year on average to find a comparable solution to the ones found by the evolutionary algorithm in a fraction of a minute.

The primary reason for this is that there is a very high branching factor in the data-set. As can be seen in Table 21, the search space balloons out in the 3-6 asteroids region. There are millions of potential routes with just the minimum length of 2 and billions of routes with the 5-6 asteroid visits we see in our good solutions.

Based on the average number of asteroids explored by the Monte Carlo, we can estimate how long routes it usually finds. An average of 30 asteroids, as seen in Figure 37, for 12 spacecraft would suggest that each on average explores 2.5 asteroids. This is significantly lower than any average we have with evolutionary runs, which typically manage an average of at least 4 asteroids explored per BEE and usually closer to 5. This is also slightly lower than the distribution of asteroids would suggest, even when the amount of overlap (Figure 33) is taken into account. It is, however, close enough that we can attribute the difference to having only made a million samples from a solution space with over a googol possibilities (around 10^{120}).

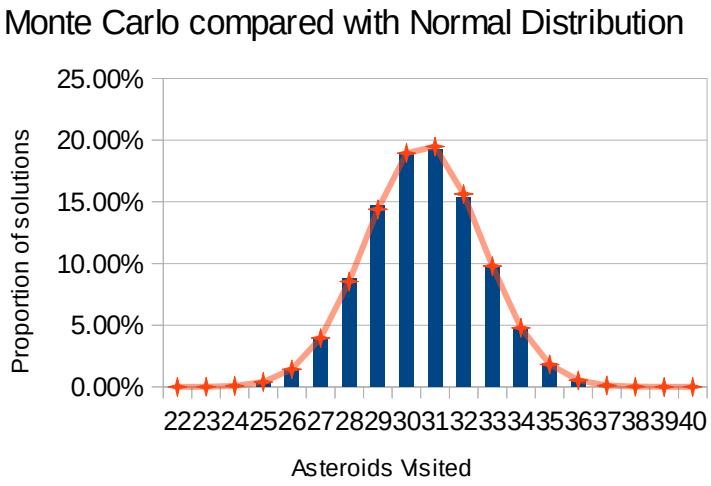

Figure 37: Monte Carlo results compared with a Normal Distribution with a mean of 30.61 (the Monte Carlo's average) and a standard deviation of 2.01 (same as the Monte Carlo).

It is worth noting that the method used by the Monte Carlo simulator to find random routes is identical to the one used to generate the initial population for the evolutionary algorithm. Which proves that the evolution is having a positive significant effect on the solution quality, since we routinely reach over 55 unique asteroids.

8.4 Full Communication

The Full Communication algorithm managed to not only explore more asteroids on average than Monte Carlo, it also achieved a higher average return per asteroid. The evolutionary algorithm is able to selectively choose routes that visit several high value asteroids and classes, while Monte Carlo relies on random chance to provide utility.

We hypothesised that our Full Communication algorithm would find solutions in the upper five percentiles of the original solution space, as described in section 1.3.1. The algorithm turned out to not only satisfy the hypothesis, it far exceeded it. It lies well within the upper trillionth of solutions which could be found by Monte Carlo. A major contributor to this is the difference between the hypothesised and measured distribution of solutions, as seen in Figure 38. It turns out that the distribution is tightly centred around a range of utilities far below the optimum, leaving plenty of head room for search strategies to improve upon the statistical expected value.

Comparing the Full Communication scenario to the Stigmergy and No Communication alternatives shows a striking difference in strategy. Figures 31 and 32 show that the Full Communication algorithm produces routes of lower utility that visit far fewer asteroids than the other two algorithms, while the scientific utility per asteroid is higher.

One contributor to this effect is the function for class utility. The function reduces the reward for every visit to a member of the class. Due to the abundance of mainly three classes, more asteroids visited would largely mean less class utility per asteroid visit. The result is that visiting less asteroids is likely to increase the average utility of the remaining asteroids while reducing overall utility of the mission.

The Full Communication scenario allows the spacecraft to coordinate their plans to optimise the return of class utility for the combined mission. Spacecraft able to visit rare classes can prioritise

them over common classes reachable by many others to increase overall asteroid and mission utility. This also applies to the Stigmergy scenario, but to a lesser extent because the spacecraft are not able to co-evolve a common mission plan.

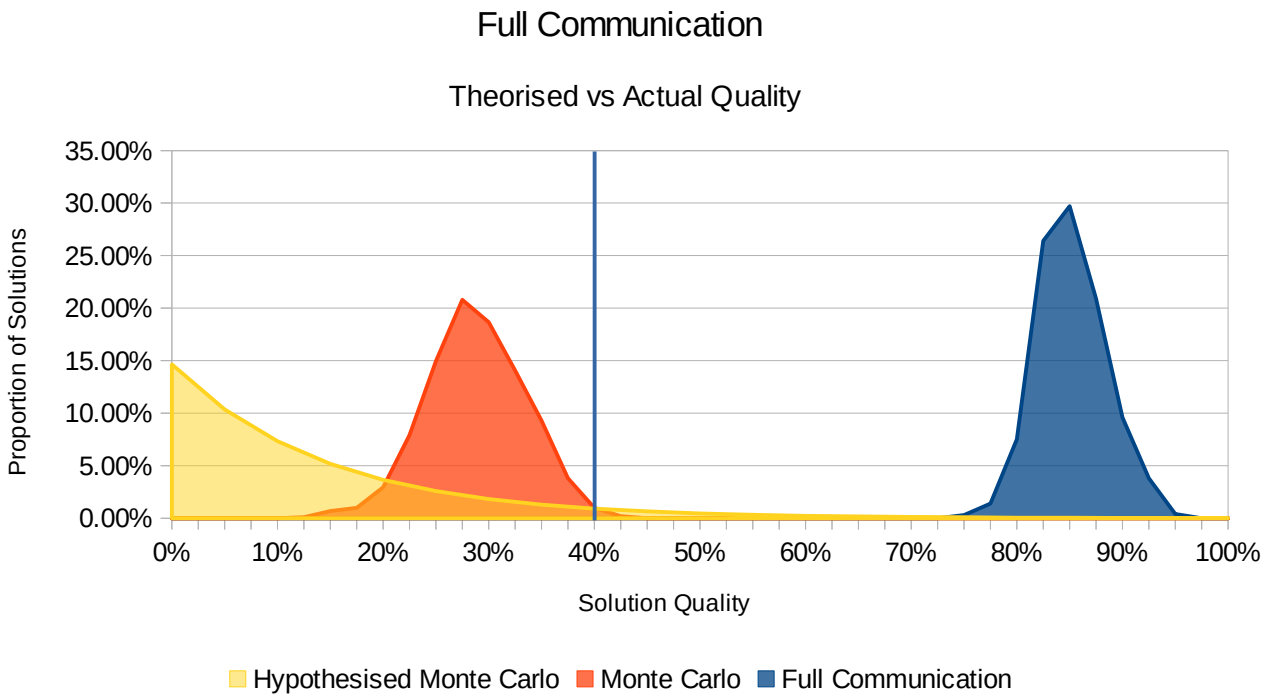

Figure 38: Shows the hypothesised and measured solution distribution versus the distribution of the Full Communication algorithm. The vertical line shows approximately where the upper 5 percentiles of the hypothetical distribution begins

Three variations on the Full Communication algorithm were run, as per 6.4.1. Both doubling the number of generations and doubling the number of co-evaluations improved the results of the algorithm, as can be seen in Table 14. The two other scenarios were still able to achieve higher utilities on average, however (Tables 16 and 18). Full Communication seems to have a strategy that is less optimal than that of Stigmergy and No Communication.

Based on the differences in the amount of overlap, it is likely that the Full Communication has prioritized reducing overlap over achieving higher utility. This suggests that the evolver is not able to get past the reduction in fitness from overlap to find the longer route.

Another, more unfortunate possibility, based on the similarity between the Full Communication overlap and that of Monte Carlo, is that Full Communication is still effectively picking random routes. This would explain why it seems to have the same amount of overlap as Monte Carlo gets through random chance. The much higher utility achieved would seem to preclude this theory, however, as the evolutionary algorithm is clearly finding routes better than any which can be randomly discovered.

The co-evolution of Full Communication means that a BEE keeps routes that perform well with the routes of other BEEs. It does not differentiate between those that perform well because of low overlap and those that perform well simply because they by chance are matched with high utility routes. Just as low utility routes can be kept because of this, the opposite is also true; some good routes can be eliminated due to unfortunate evaluations. The random effect co-evolution has on the estimated route utilities may have an adverse effect on what Full Communication is able to achieve.

Doubling the number of generations helped Full Communication's performance, as did doubling the

number of co-evaluations per route. Increasing the number of generations does not provide a significant benefit if the population has already converged on local maxima, suggesting that the algorithm could not quite achieve optimal performance with 100 generations. The improvement was only marginal, however, meaning that a full doubling of the number of generations was not necessary.

Increasing the number of co-evaluations allows the utility of each route to better be estimated. Evaluations are performed with a wider range of routes, reducing the effect of random chance. Full Communication achieved higher utilities on average with additional evaluations, but as with additional generations, it only increased performance slightly.

8.5 Stigmergy

The original hypothesis (see section 1.3.1) was that Stigmergy would not be able to reach the same performance level as Full Communication because of its reduced level of communication. While Full Communication should allow the spacecraft to act as one, Stigmergy only provides infrequent updates on the current and past activity of the swarm. Reducing the communication was thought to increase the level of overlap in routes and by that, reduce the overall utility of the mission. The results do not correspond with this hypothesis. Stigmergy, with its initial No Communication planning, is able to outperform the Full Communication scenario.

One can see from Table 16 that the greedy variant is the best Stigmergy alternative. The greediness has the spacecraft attempt to maximise individual scientific return. They still receive and use information about the number of previous visits to the asteroid classes, which updates the perceived utility of asteroids and may alter plans to avoid unwanted overlap. However, as Stigmergy and No Communication seems to have very similar levels of overlap regardless of heterogeneity, it would seem that the BEEs are unable to utilize the available information to avoid areas of overlap.

Alternatively, the BEEs are aware of the overlap, but avoiding it would reduce the total utility of the mission to the point where it is not worth it. In this case, accepting some overlap during the mission could perhaps allow the spacecraft to explore two separate high value targets accessible from a single easily reached asteroid.

In this case, it could be that working from a No Communication baseline is a blessing in disguise for the Stigmergy evolver. The Full Communication evolver, which can detect overlap instantly, may never be able to make the leap past the overlap to reach the peak utilities of the other evolvers. The Stigmergy evolver thus starts at a higher utility, where reducing overlap would be a step backward.

8.6 No Communication

The results in Figures 31 and 32 show that No Communication in no way lags behind the other two scenarios. The hypothesis was that the performance of No Communication would suffer due to overlap, where spacecraft generally visited the same asteroids and asteroid classes. The hypothesis further stated that the communication provided by Stigmergy and Full Communication would reduce and eliminate this problem, respectively.

Instead, results show the opposite; the more information is available to the swarm, the worse it performs. Only Stigmergy is able to achieve mostly the same average utility as No Communication, and it uses an initial No Communication evolution to provide a base for further improvement. This could be an artefact of the algorithms involved, where the additional information confuses the evolutions and keeps it from finding the highest utility routes.

No Communication has the disadvantage that it cannot detect or handle overlapping plans. This is in no way a problem if the overlap is negligible. A general rule of thumb in computer science is that

the more specialised an algorithm is, the better it performs if its assumptions are met at the cost of reduced performance if they are not. The negative correlation between level of communication and performance found in the result may have its root cause in this.

The results may depend on the Transfer Map consisting mostly of three asteroid classes with equal values. It is possible that this distribution does not produce noticeable overlap by not encouraging spacecraft to visit the same small subset of asteroids. There is still a question of intrinsic asteroid values, ranging linearly from 1 to 10. Spacecraft should have been encouraged to target only high value asteroids, but it seems this is not providing enough pressure to produce detrimental amounts of overlap.

Other configurations may see more overlap. There could be significant amounts of overlap if the asteroids followed a different value distribution or the asteroid class distribution had some classes be both rare and valuable. For example, the amount of asteroids with values above 9 could be limited to 1% rather than 10% and a fourth class with twice the value and a fraction of the frequency of the big three could be added. High value members of that class should then be targeted by every BEE able to reach them, causing overlap.

8.7 Compared to APIES

The original APIES plan, as described in section 2.2, involved 19 BEEs in a net formation around a central HIVE. It would have flown by a hundred asteroids over the course of 6 years. Our best solutions would have 12 BEEs orbiting just over 61 asteroids over the course of 20 years.

The biggest difference between our variant and the original APIES is the decision to orbit each asteroid, rather than just do a flyby. With a flyby, the spacecraft would pass by the asteroid at a significant velocity, while an orbiting spacecraft can study the asteroid for months on end, until the transfer window to the next target opens. This decision was explained in chapter 4.

There is no easy answer to whether a detailed examination of 60 asteroids is better than a quick scan of 100, as it depends on what one hopes to learn from the mission. Being able to orbit the asteroid for months on end would allow the BEE to make very detailed examinations of the shape of the asteroid, its mass, and the spectrographic reflections of the surface layer (a good indicator of composition). A flyby, on the other hand, would only be able to gather data for 14 hours, and visual imagery for only 15 minutes. As only half the asteroid would be pointed at the BEE during those 15 min, a large portion of the surface would go unexamined. The BEE would still be able to provide a strong estimate of the shape of the asteroid, a good estimate of the mass, but only a limited examination of the surface layer.

As such, which approach would provide the best return depends on the end goal. For asteroid deflection missions, the density (determined by the shape and the mass) is the most important, as it is a strong indicator of whether the asteroid is a low density “lose pile of rubble” or a high density “dusty solid”. For asteroid mining, on the other hand, the exact composition of the minable surface material is important for determining the economic value of the asteroid.

The original APIES mission would be better as a preparation for potential asteroid deflection missions, as it provides exactly the data they need, and for a larger set of asteroids. Our variant would be better for asteroid miners, especially as they could assign a high scientific value to potentially minable asteroids a high value to classes known to consist of useful materials. A major strong suit of our plan is the ability to direct the BEEs towards certain targets, rather than examining whatever asteroids happen to pass through the net.

The difference in the time required by the missions is noteworthy. The original APIES mission was only 6 years long, with potentials for expansions if the swarm is still operational after that period. Our mission time is significantly longer, which would mean more expenses for maintaining a

mission control centre on Earth. It also presents a bigger engineering challenge, as the BEE's would need to be rugged enough to have a good chance of surviving the length of the mission. All in all, this does mean that the APIES mission is likely cheaper, despite requiring 7 more spacecraft than ours.

8.8 Value of Communication

The original APIES plan uses unique intercept zones for each BEE, with some ambiguity of which of three BEEs to perform an intercept of asteroids passing between the individual zones. For the most part, the ambiguity can be avoided completely by not intercepting outside the zones, or manually assigning the ambiguous areas to specific BEEs. Although this might not be an ideal solution utilitywise, it would at least eliminate overlap. Because of this, the plan was modified to increase the need for communication. The BEEs were required to travel to asteroids instead of moving within their own intercept zones with the occasional ambiguous intercept. An identical starting point and state would aggravate the problem.

Unfortunately, it would seem that the modified plan has even less need for communication, as the little overlap we see coincides with the highest scientific return. As stated in Appendix B, our solution space is a subset of what we would have had if there was enough time, and the worst solutions were eliminated first. And, by the time swarms of spacecraft are sent to the asteroid belt, the amount of new asteroid discoveries will most likely have dwarfed the amount known at the time of writing. In the few months since we first retrieved the “astorb.dat” asteroid database[22], almost ten thousand new asteroids have been discovered, an average of roughly 80 new asteroids a day. And the rate of discoveries is rapidly increasing as better telescopes and computers become available.

Even with this heavily reduced solution space, the amount of overlap in spacecraft plans seems trivial to the point of having little to no influence. The initial branching factor seems high enough for the handful of spacecraft to disperse and mostly find their own routes in the asteroid belt, making communication all but superfluous. The goal of communication is after all to allow the spacecraft to detect and avoid overlapping routes and thus avoid duplication of effort, all while they pursue the same high value routes. The sheer size of the asteroid belt seems to be enough to all but eliminate the need for such measures. Actively avoiding the little overlap there is may also have a greater adverse effect on the exploration than what can be earned with the saved propellant.

Considering the substantial cost of launching a mission to the asteroid belt, taking measures to avoid a rare but potential problem is still justifiable. There may still be a need for communication to control overlap with different algorithms, algorithm parameters, transfer maps, asteroid and class values, or mission parameters. Detecting and forestalling overlap before it causes a utility loss would therefore be desirable, especially to ensure that they do not make the same opening transfer.

9 Conclusion

We set out to find out how important communication is to the efficiency of an exploration swarm of spacecraft operating in the asteroid belt. By comparing evolutionary algorithms with access to different levels of information about the activities of their peers, we could largely isolate the effects of communication from other implementational factors.

Communication would be vital for missions involving a large number of spacecraft or a limited number of target asteroids. However, the asteroid belt covers a vast area and contains hundreds of thousands of asteroids. As no currently planned mission would involve more than 20 spacecraft, it seems individual spacecraft can get by with attempting to maximise their own contribution to the scientific returns of the mission without considering the others.

In fact, we find that the costs associated with taking sub-par transfers to avoid overlap actually reduces the amount of exploration done overall. Both with no communication at all and with limited (stigmergy) communication, our evolvers performed better as purely greedy agents than with more considerate behaviours. With co-evolution, we have found that it takes significantly more computing power to reach the same level of results which can be obtained by purely greedy agents.

We have implicitly assumed that a wide sampling was preferable to investigating specific targets. With such a wide range of options it is not too hard to see why overlapping routes is unlikely to be a problem. Even without a complete database of potential transfers, we had over three thousand potential transfers out from our initial location.

That is not to say that there was no overlap. But evolvers with a high degree of overlap do not receive poor results in terms of unique asteroids explored or the scientific return of the mission. Whether or not a specific overlap incident is harmful for a given mission is not as clear cut as we initially had thought, and would have to be carefully considered by future mission planners.

9.1 Research Questions

In section 1.3, we introduced a number of questions we wanted to answer with this thesis. Having completed our experiments, we are now capable of answering these:

RQ1 Full Communication was on average able to visit 48.7 asteroids and achieve a scientific return of 670.8, compared to Monte Carlo's 30.6 asteroids and 384.7 science. Full Communication reliably produces mission plans far better than any found by Monte Carlo, and thus far beyond the hypothesised upper five percentiles.

RQ2 Stigmergy was on average able to visit 55.6 asteroids and achieve 694.2 points of scientific return. There does not seem to be any performance loss whatsoever for the particular parameters used in our experiment. On the contrary, there seems to be a major performance gain when reducing the degree of communication within the swarm. The stigmergy method easily outperformed co-evolution.

RQ3 Heterogeneity does not only seem sufficient to ensure reasonable efficiency, it seems unnecessary. Homogeneous agents did not suffer an efficiency loss, even without communication. Homogeneous No Communication was able to visit 55.8 asteroids and gain 694.5 points of scientific return. No other method produced better results.

9.2 Applications

To the best of our knowledge, no previous attempt has been made to map the transfer windows of the asteroid belt at this scale. Our present database is somewhat limited; it is incomplete and was built specifically to examine the potential transfers out from the HIVE's initial position. But the

transfer map generator we have produced and tinkered with throughout the development process can be of use for any multi-asteroid exploration mission, not just a modified form of APIES.

As such, we have chosen to make our source-code available online as a git repository,* so that other researchers can benefit from our work. As described in Appendices B, C, D, and E, we were caught off guard by the sheer magnitude of the problem, and spent a significant amount of time working around it. As a result, the transfer map generator has more optimisations and development than was originally intended and is considerably more capable as a result.

Our actual results may potentially be of some use to mission planners. We have proven that it is possible for a patient mission planner with access to a wide range of targets to explore multiple destinations with very little propellant requirements. Our average BEE explored 4-5 asteroids with 3km/s of delta v. For comparison, NASA's Dawn mission had three times the delta v and explored only two asteroids (4 Vesta and 1 Ceres).

The Dawn mission was planned with specific targets in mind, which has been the traditional way to design space probes. As discussed in section 1.1 however, that is changing with a new class of cheaper autonomous spacecraft being developed. This thesis does demonstrate the capacity of evolutionary algorithms in autonomously planning missions, as well as establish that it is certainly possible to explore a wide range of asteroids with only limited propellant.

*Available at: https://github.com/yrgx1/astorb-transfer-map-generator

10 Future Work

Our principle limitation when it came to this project was the lack of calculation time. With access to a more powerful computer or more time to run the calculations we could relinquish the time saving measures we had been forced to implement.

First of all, our search of the asteroid belt held a number of innate limitations. We observed that the number of potential transfers grew exponentially as the maximum allowed delta v was increased. If we could increase that limit from 1200m/s to the full propellant capacity of our modified swarm, the search space would much more closely resemble the actual environment. Similarly, we could expand the time limits even further, allowing BEEs to wait longer between transfers.

In addition to being useful for further experiments, such a map of asteroid transfers would have intrinsic value on its own. To the best of our knowledge, a complete statistical analysis of asteroid belt transfers has not been done and it would certainly be interesting, if time consuming, to produce an all-to-all transfer database for main belt exploration.

We could also go from simply transferring between asteroids to the more computationally complex option of making flybys of asteroids. This would only require the BEEs to make one burn, but would mean every transfer would leave the spacecraft in a unique flyby orbit rather than in orbit around an asteroid. As a result, our branching factor would be massively increased and the reduced propellant requirement would also increase the search depth. So it would be a significantly larger search, which would allow for more exploration and a fairer comparison to the original APIES plan.

With sufficient access to computing power, we could also attempt to replicate a larger mission, which would have significantly more potential for overlap and would require significantly more communication to coordinate. The PAM/ANTS[3] mission proposed by NASA's Goddard Space Center would make an excellent case study. With a thousand spacecraft operating in a hundred teams, it would be a significantly larger project than APIES, and thus more likely to run into the issue we have investigated.

ANTS would also feature multiple types of specialised spacecraft with different types of instruments. As each scientific instrument would likely be able to make new discoveries about an asteroid, each class of spacecraft would only have to worry about overlap from within its own class. It would be interesting to see if the AI would naturally produce teams of spacecraft with a full set of instruments, or find a solution other than the one envisioned by NASA.

Given time to adjust the code, we could also change the way our evolution works. Thus far, the evolutionary algorithm has terminated after a fixed number of generations. This is inefficient if the solution converges quickly and may be sub-optimal if the solution has not yet converged. Implementing a convergence test could make the evaluation faster or at least provide better solutions. We could investigate the effects of increasing or decreasing the number of crossover attempts per generations or vary the size of the base population.

One of the things we did make some preliminary investigations into was using an indirect genome encoding (documented in section G.2). The results were not promising, but our investigations were entirely done using a Transfer Map from the initial 6 year search, which was flawed in many ways. It would be interesting to examine it closer with the extended time frame, in which we managed to produce significantly longer routes.

Stigmergy's reliance on on-board computational power for real time planning would have been interesting to investigate. The code would have to be adjusted based on how much computing power the spacecraft can devote to evolutionary algorithms, perhaps by limiting the number of generations each BEE can run depending on when it needs to make its next transfer.

Further experiments with other behaviours would also be interesting. In retrospect, a homogeneous Stigmergy based swarm might be better off with the Proactive (avoid future overlap) and Considerate (avoid past overlap) than with the pure Greedy behaviour we actually tried. This should avoid what little overlap we have, and thus be strictly better than the No Communication solution it is based on.

Our Full Communication code did not perform as expected, which may be due to the inherent variance in the co-evolution evaluation method. In which case we could try a hybrid solution, which gets half its fitness from co-evolution and the other half from the equivalent of a Greedy No Communication evaluation of the genome. This would pressure it into producing more utility per BEE while still receiving enough information about the other plans to improve the overall solution.

Another possibility, based on the observed differences between Full Communication and our other results, is that there is not enough pressure towards creating longer routes. Adding the total number of asteroids explored to the fitness of a co-evolution evaluation would give slightly more fitness to longer routes, which may be enough to push it into finding the better solutions found by the other evolvers. As the scientific return tends to be about ten times as high as the number of asteroids explored, this would result in the asteroid having about a 10% weight in the fitness function.

Another interesting option would be to run Stigmergy based on an initial set of plans produced by a Full Communications run. It would be interesting to see whether this would behave any differently than the original, No Communication-based, solution. Another option would be to develop the original plans using the Stigmergy code itself, which would effectively mean a pure greedy solution as all of the other Stigmergy behaviours rely on information not yet available at that point.

The assignment of classes and class values to asteroids, which we did in section 5.3, would certainly have been done differently if we had known how few asteroids there are with known classes. A potential improvement would be to occasionally assign one of the less common classes when an asteroid's actual class is unknown, in order to ensure that the distribution of classes better mirrors reality.

We could also experiment with what happens if a small subset of asteroids receive a much higher priority. We would expect this to case higher levels of overlap, but we have not had the time to examine it thoroughly yet.

Appendix A Languages and Libraries

During discussion of the implementation described in chapter 5, we faced the question of which programming languages and third party libraries to use. This appendix quickly summarises the reasoning behind which programming languages to use and which libraries were used within the Transfer Map Generator described in section 5.1.

A.1 Choice of programming language

Choosing which programming language to use is always an important decision. Given our experience, there were three natural choices: *C++*, *Java*, and *Python*. The three have different strengths and weaknesses. Each component of the complete system could easily be developed in its own language, as communication was designed around passing files. Matching component with language would allow us to play to the strength of each language while largely avoiding their weaknesses.

The first component was the Map Generator, which would produce a map of transfer windows. Another component would assign a persistent scientific value to each asteroid. It would also assign classes to those which have not yet been classified. The files from these components would be used by the AI Simulator to produce the final results.

The strength of Python is that it has little structural code and extensive standard libraries. The downside is that the lack of structural code makes large programs difficult to maintain. Although reasonably performing code can be written in Python, its strength is in input/output oriented computing. This makes the language a natural choice for assigning values and classes to asteroids, but a poor one for the more computationally challenging components.

C++ can produce the best performing code of our three options. Although the performance of Java is comparable to that of general *C++* code, *C++* has the advantage of also allowing high performance *C*-like code where necessary. The language is mostly backwards compatible with *C*, allowing the use of both *C++* and *C* libraries. If high performance code or libraries are needed, *C++* is be the natural choice. Unfortunately, *C++* arguably requires more skill and effort to produce the same quality code as Java. It generally requires more code and is far more error prone when writing *C*-style code due to its explicit handling of memory.

Unfortunately, only one member of our team had sufficient training to quickly develop and maintain *C++* projects, so we wanted to limit the size of any such undertaking. The Map Generator was meant to be a small and quick component to develop, with a need for third party libraries and high performance. The choice thus fell on *C++*.

The main component of the system was the AI Simulator. Because it would have to be developed by both team members, *C++* was not an option. Performance was still a concern and the program would be relatively sizeable. The choice thus fell on Java, as it should handle both traits better than Python.

A.2 Libraries Used

The first step to a coding project is generally to see how much of it other people have already written. With this in mind, the first thing we did was to search for libraries with the functionality we required, most notably astrophysics calculations and implementations of the Nelder-Mead technique.

We looked at writing the Map Generator in *C++* to take advantage of its excellent performance in order to minimise the time required to generate the Transfer Map. Secondly, taking advantage of

parallel computation, the run time would effectively be reduced by a factor of 4 on the available quad-core CPU (a 2.3GHz *Intel Core* i7-3610QM with 4 physical cores, 8 logical cores). This can easily be achieved in *C++* using OpenMP directives[41].

The European Space Agency's Advanced Concepts Team (ACT) has produced a very successful library for Keplerian mechanics called PyKEP[42]. While it is ostensibly a Python library, the majority of PyKEP is written in *C++* for efficiency. As we are using only *C++* for the asteroid belt model, we simply ignore the python bindings. PyKEP includes a rapid solver for Lambert's problem (described in section 2.1.1) and translations between orbital elements and Euclidean position/velocity vectors at a given time.

We have looked for a viable implementation of the Nelder-Mead technique, described in section 3.7. But, as explained in section 5.2.2, the free and open source implementations we found were poorly documented or otherwise unusable. After two days of searching in vain for a usable implementation we decided that it would be easier to implement our own.

Appendix B Optimisations

This appendix covers the problems faced during implementation of the Transfer Map Generator in more detail. Section 5.2 summarises the end result and main difficulties described in the following sections. The limitations put on the Transfer Map Generator influence the conclusions drawn regarding the research questions. It is important that the reader understands these limitations and why they were necessary for the completion of the project.

The appendix covers the three aspects most relevant to the end result. It first describes the process of simplifying the search required to find sufficient transfer windows between asteroid pairs, at the cost of reduced accuracy. Second, it describes how the problem size was reduced by removing difficult to reach asteroids using a filter. And third, it describes how these two measures were not enough, and which further reductions were taken in order to save on run time.

B.1 Initial Run Time Estimates

The map generator needed to take in a database of asteroid orbits and compute all transfer windows for asteroids that the BEEs could reach. A naive solution would be an all-to-all search for all asteroids in the database. The complexity of such a search would be:

$t_{runtime} = O(n^2)$, where n is the number of asteroids in the database.

The complexity of such a search would be prohibitively large unless n or the time needed to find the transfer windows from one asteroid to another ended up being trivially small.

We had found a comprehensive database[22] of every asteroid ever observed, compiled by Dr. Edward Bowell at the Lowell Observatory. The database contained approximately 700,000 asteroids. We expected to be able to eliminate asteroids outside the mission area, but several hundred thousand would likely remain. We estimated that the time needed to find transfers between two asteroids, c, was in the hundreds of microseconds to milliseconds range. This gave us a rough run time estimate of:

$$t_{runtime} \approx c \cdot n^2 \approx 1\,ms \cdot 500{,}000^2 = 250{,}000{,}000\,s \approx 8\,years$$

Even with a c of 100 microseconds, the model would not be completed in the few weeks available. The complexity of the problem clearly had to be reduced.

A depth first search from the HIVE with a depth limited by the delta v available to the spacecraft would reduce the complexity to a reachable-to-all search. If, as we suspected, only a fraction of the asteroids would be reachable, this would reduce the run time greatly. The complexity of the search would then be:

$t_{runtime} = O(r\,n)$, where r is the number of asteroids reachable from the HIVE.

Looking at the previous estimate, even if the number of reachable asteroids were as low as 10% of the total, the run time could still be in the order of months.

Designing our code to take advantage of modern multi-core processors would allow the code to be sped up by a factor of p, the number of physical CPUs available. Hopefully, this would be sufficient, but there was no way to tell before we had implemented the code. Our upper limit of c, assuming a maximum run time of one month, p=4 (quad core processor), and an estimated r of 10% of n, was:

$$t_{runtime} \approx \frac{c \cdot r \cdot n}{p}$$

$$c \approx \frac{p \cdot t_{runtime}}{r \cdot n} \approx \frac{4 \cdot 1\,month}{0.1 \cdot 500{,}000^2} \approx 400\,\mu s$$

There was unfortunately no way of knowing the precise values of *c*, *r*, and *n* until we had implemented the search. Our original estimate of *n* was quite close; roughly 650,000. However, the constant *c* was off by orders of magnitude. We had expected *c* to be roughly 1ms, give or take an order of magnitude, but measured it at roughly 15 minutes. This meant that any attempt at estimating the value of *r* would take days, if not weeks. With the updated values of *n* and *c*, and the initial guess of *r=0.1n*, the run time was estimated to:

$$t_{runtime} \approx c \cdot r \cdot n \approx 15\,minutes \cdot 0.1 \cdot 650{,}000^2 \approx 1{,}1 \quad million\,years$$

This meant we would need a total speedup of more than ten million to have a chance at solving the problem. Most of the development time was spent optimising the performance of the code to a point where the constant was somewhat within acceptable limits, with an estimated single thread speedup of 300,000 and a parallel speedup of 4.

B.2 Reducing the Constant

The single most resource intensive code in the asteroid belt model generation was the Lambert's problem solver, which would be run quintillions (10^{18}) of times in the course of the full program. Profiling revealed that almost 100% of the run time was spent solving Lambert's problem. Improving the run time of the Lambert's call itself was not practically possible as the PyKEP implementation had already been extensively optimised by the developer. Naturally, our efforts have instead been focused on reducing the required number of calls as much as possible.

Our first optimisation was to use a triangular search area, ensuring that all generated transfer windows fit within both departure and arrival time constraints. Previously, we pruned invalid transfers only after they were generated. This halved our run time.

Secondly, our original version of the code divided the solution space into triangular sections and ran Nelder-Mead with each triangle as an initial simplex for the algorithm. This proved very computationally expensive, as it meant running Nelder-Mead a million times per asteroid pair. Some quick sketching revealed that the same initial points could all be investigated using one sixth the number of simplexes, if a carefully chosen sub-set was used (as seen in Figure 39).

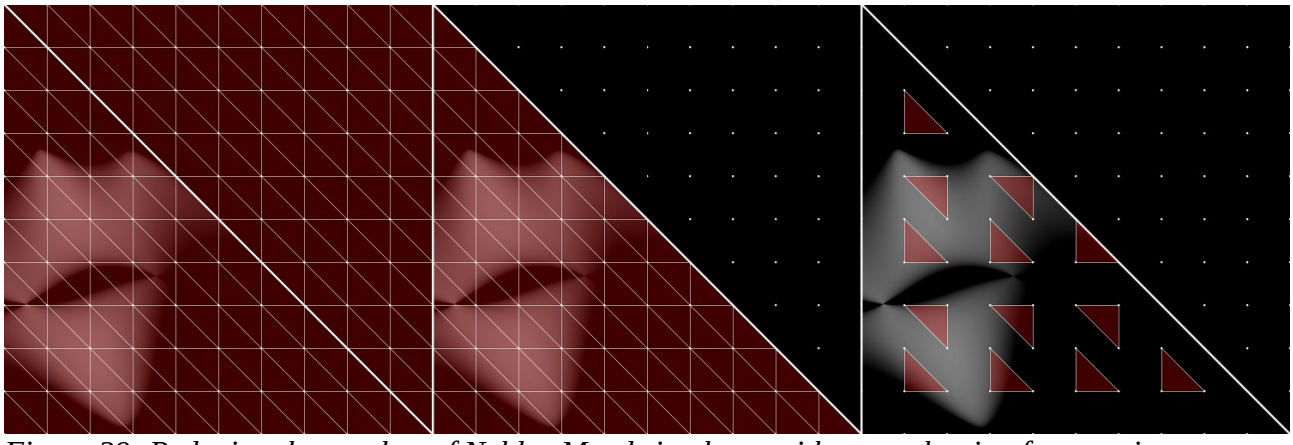

Figure 39: Reducing the number of Nelder-Mead simplexes without evaluating fewer points.

Furthermore, since we were effectively producing a grid of search points in the process of generating initial points for Nelder-Mead, we might as well utilize that information in picking where to search. By excluding simplexes where every initial value was above a chosen threshold, we were able to find all the same transfer windows while reducing search time by a significant but

varying amount, (typically between 30-70%).

The Nelder-Mead technique iteratively performs one of several transformations on a triangular simplex depending on the measured fitness at the vertices. One of these transformations, called reflection in the algorithm, flips the triangle across a carefully chosen point. As long as only reflection transformations have been performed, the simplex still has all its vertices on points in our search grid. We had already calculated the delta v costs for these points. In addition, in rare situations (roughly 4% in one test case), two or more simplexes are reflected such that they have all the same points. Because Nelder-Mead is entirely deterministic, this would result in them climbing to the exact same point.

We chose to implement a reflection only version of Nelder-Mead. This algorithm terminated whenever a full Nelder-Mead would perform another transformation than reflection. Once every initial simplex had been run through this algorithm we could remove duplicates and run a full Nelder-Mead on the remaining simplexes. Not only did it save us from the 4% duplicated work load, it also reduced the average number of iterations before Nelder-Mead converged by about 3, as the initial reflections were done in advance without the necessity of solving Lambert's problem for each of them.

A later innovation on the limited Nelder-Mead was to perform reflections in a wider set of situations. In essence, we performed reflections whenever this led to a better initial triangle. This increased the savings, cutting out a third of all Nelder-Mead runs required.

We have also investigated varying the size of the search grid. Run time scales quadratically in the resolution of the search grid, so we naturally wanted to make it as sparse as possible. On the other hand, if the grid is too sparse we start missing launch windows entirely. With some experimentation, we were initially able to reduce it to 50x50 (a resolution of approximately 45 days).

The limiting factor was that we excluded all simplexes where none of the points of the simplex were within the delta v limit of the search. As the resolution was reduced, marginal transfer windows started slipping through the cracks. Keeping every simplex allowed us to reduce the resolution to 12x12 (6 months) without missing any transfer windows.

With the reduced resolution, smarter selection of initial simplexes, and the reflection only Nelder-Mead function, we managed to reduce the number of Nelder-Mead runs by a factor of 150,000. In addition, the convergence time for the Nelder-Mead was reduced by a factor of about 6 (most of this from the reflection only code), reducing the total Lambert calls down to trillions (10^{12}).

B.3 Problem Size

Having cut the *c* constant as far as we could without losing accuracy, we started looking at the other factors in the run time of the code. We wanted to be sure that we kept every reachable asteroid, *r*. However, searching for transfers to asteroids we could not possibly reach provided no benefit. The goal was to cut *n*, the size of our database, down to as close to *r* as possible.

There were 676,327 known asteroids at the time of writing[22]. As this included asteroids as far out as Jupiter and others closer to the Sun than Venus, chances were we could only reach a small number of these. Our initial estimates, extrapolated from experiments with a small sub-set of asteroids (typically 0.1%-1.0%) was that there were about 10,000 to 50,000 reachable asteroids in total.

Our first pass filter was to remove asteroids whose location was not known with high precision. This was used as a first pass filter largely because the database[22] we used already included warning flags for uncertainties, so it was very easy to implement. This first pass reduced the number by 32,032, a mere 5%.

The second pass was slightly more complicated. With the limited delta v available, there were

limitations on how much the BEEs could alter the potential energy of their orbits. In the standard Keplerian elements (section 2.1), the semi-major axis is an indicator of the potential energy of the orbit; how high up in the Sun's gravity well it is. As a result, we knew there would be a limited range of semi-major axes that the BEEs would be able to explore. We did not, however, have any easy way of determining that range. Similarly, changing ones inclination is a very delta v intensive manoeuvre but it is not trivial to determine how expensive it is.

However, it would be possible to search the database in order of increasing and decreasing semi-major axis, and return the first asteroid with an available transfer window from the HIVE. Similarly, we can order them by inclination and do the same. This reduces our problem size linearly, but risks removing reachable asteroids if not done with care. The maximum delta v for the search should be slightly higher than the capacity of the BEEs to be certain that no reachable asteroids are lost. However, the reachable asteroids lost are the ones most difficult to reach, which would likely have low priority in the evolutionary algorithm. With a delta v limit of 1700m/s, the database's size was reduced down to a third (212,077 asteroids).

B.4 Heavy Optimisations

Unfortunately, even reducing the run time by a factor of approximately 300,000 was not enough. We were unable to reduce c to less than roughly 3ms. With parallel execution on four cores, we were able to effectively reduce c to approximately 700µs, still almost an order of magnitude more than our available time allowed.

With at most a few weeks available for computation, this alone warranted further and heavy optimisation. Unfortunately, the increase in BEE delta v capability, from 1700m/s to 3400m/s, meant more asteroids could be reached. This increased the size of n to roughly 450,000, changing the estimated run time:

$$t_{runtime} \approx c \cdot r \cdot n \approx 700\,\mu s \cdot 300{,}000 \cdot 450{,}000 \approx 1.4 \quad years$$

We first looked at the resolution of the Nelder-Mead search grid. Previously, the resolution was 12x12, as lower resolutions missed some transfer windows. However, reducing the resolution would provide a significant speedup. After experimenting with different resolutions, we settled on 9x9, which retained virtually all the transfer windows while halving the constant.

Secondly, the increased range of the BEEs allowed them to reach asteroids far outside the mission area. With the filtering employed previously, most asteroids in the database were included, about 450,000 out of the potential 650,000. Exploring these asteroids would not only require high amounts of delta v, but would take the spacecraft far away from the main asteroid belt where most asteroids reside. Returning to the main belt would require another expensive transfer. By filtering asteroids not based on the 3400m/s delta v limit, but on the previous 1700m/s limit, the number of asteroids was reduced back to the original number of roughly 210,000. The effect on the mission was to eliminate likely inefficient transfers out of the main belt, forcing the spacecraft to make overall better choices.

Although more than half the asteroids were removed, the number of reachable asteroids was not impacted the same way. The asteroids removed by the filtering were, after all, the ones most difficult to reach. This left us with an extreme branching factor, where spacecraft were allowed to spend most of their delta v travelling to difficult-to-reach asteroids without much range left.

We similarly modified the code to only allow transfers with a cost of at most 1200m/s. This ensured that BEEs would not be allowed to spend all their range exploring one or two asteroids, and instead forced them them to make overall better choices. At the same time, these difficult-to-reach asteroids would not be included unless there was a route to them conforming to the 1200m/s limit. This would, we hoped, eliminate a large fraction of the marginally reachable asteroids.

Both of these restrictions on the choices of the BEEs were motivated by a need for a reduction in run time of the map generator. The run time of the AI Simulator would also likely benefit from a smaller map of the asteroid belt, and thus a smaller complexity for the evolutionary algorithms. However, removing the choices least likely to be considered good by the AI could influence its behaviour and the measured value of communication.

Before starting the full scale map generation, we made a final estimate of its run time with the following assumptions: *r of 50,000-100,000* and *n* of 210,000 asteroids. This provides us with lower and upper estimates of the run time:

$$t_{runtime,low} \approx c \cdot r_{low} \cdot n \approx 350\,\mu s \cdot 50{,}000 \cdot 210{,}000 \approx 40 \ \ days$$

$$t_{runtime,high} \approx c \cdot r_{high} \cdot n \approx 350\,\mu s \cdot 100{,}000 \cdot 210{,}000 \approx 85 \ \ days$$

We were unfortunately never able to know for sure whether the upper estimate was reasonable. The search was ended prematurely after running for 49 days and reaching over 80,000 asteroids, as explained in section 5.7. This places us towards the end of the expected range in terms of asteroids reached, while the run time was closer to the lower estimate when ended.

Appendix C Transfer Map File Format

This appendix describes the file format of the Transfer Map throughout the project. There were two main versions of the format. The first was initially used to allow stable and reliable storage of the file while the second variant was finished. The Map Generator initially used the first version and was later converted to use the second. The following sections describe the initial file format, how the format was too inefficient, how it was improved to produce the second file format, and the relative efficiency of the two.

C.1 Initial File Format

The purpose of the Transfer Map Generator was to produce a file containing every transfer window in the asteroid belt reachable by the spacecraft. The resulting file would be read by the simulator and provide a working environment for the spacecraft.

For each transfer window, five things were necessary: origin asteroid, destination asteroid, time of departure, duration of flight and cost of transfer. The asteroids would be represented by their names; 26 character/byte long strings. The numbers were naturally represented as 8 byte doubles.

To avoid producing an unnecessarily large file, transfer windows for the same asteroid pair were grouped together. The redundant names were removed, producing a combined entry beginning with the two names and a count specifying the number of transfer windows for the asteroid pair. The time of departure, duration of flight, and cost of transfer were listed for each transfer window. The resulting entries would look like this:

```
[origin name] [destination name] [count c of transfer windows]
    { [transfer window]¹, [transfer window]², …, [transfer window]ᶜ }
```

We wanted to be able to resume calculation should the program be interrupted. Losing all progress would be acceptable if the program could be completed in a day or two. Losing two weeks of progress, on the other hand, would be a severe setback. Resume functionality would also allow us to replace the executable with an improved version without starting from scratch, as well as continue execution on a more powerful computer if one were to become available to us.

To resume, the partly completed model would simply be read back in on start up and used to speed up the search. Ideally, this would allow the progress to be resumed immediately, only losing as much time as the program was offline. New asteroid pairs would be appended to the original file as the search continued.

The resume file contained every transfer window found during the search, but not all of those were actually traversable. The cause for this is the possibility of a future route allowing an asteroid to be reached earlier or more cheaply. This could in turn allow the use of a previously untraversable transfer window. Only after the search is completed would we know for sure whether a transfer window was traversable or not.

Removing untraversable windows would not affect the results found during simulation, as they could never be part of a valid route. The map size would be reduced, however, allowing for a smaller file size, a shorter start up time, and a smaller memory footprint.

This would only be a concern if performance was inadequate, the file size or start up time excessive, or the memory footprint too large. The choice at the stage of planning was to wait and see if the amount of junk data posed a problem.

C.2 Possible Sources of File Size Improvements

The file produced by the project was initially estimated to not be large enough to pose a problem.

However, after some debate, we decided to boost the delta v limit of the spacecraft from 1700m/s to 3400m/s, as discussed in section 5.4. With twice the delta v, the spacecraft would likely be able to visit twice as many asteroids each. Since the size was proportional to the product of the total number of origin-destination asteroid pairs, this would likely quadruple the file size. Four strategies were identified and discussed, and are presented here.

The first was to compress the file using standard compression libraries. As a large portion of the file would be repetitions of the same 26 byte asteroid names, the file should be easily compressible. We chose not to do this because of the effort needed to implement the functionality compared to the proportionally small problem. The potential for introducing defects and I/O overhead solidified our position.

A second strategy was to generate a dictionary containing a mapping from asteroid names to unique 4 byte identifiers. This would reduce the file size significantly, as 52 bytes could be reduced to 8 bytes for every entry in the file, potentially halving the file size. However, adding the dictionary to the beginning of the file would require a major rewrite of the otherwise stable code.

Multiple transfer windows with the same origin and destination asteroids were already grouped in the same entry. Adding one more layer of grouping would allow all entries with the same origin asteroid to be grouped into one. While requiring one more counter, this would reduce the overhead of the names significantly, proportional to the branching factor of the search. A branching factor of 100 would on average combine 100 entries into one, eliminating 99 copies of the same string. The amortised effect would be to reduce the size of the origin field down to just over two bytes. Realistically, the branching factor could be much larger than this, further improving the reduction. Because the origin field required about a quarter of each entry, roughly one quarter of the file size would be eliminated by this method. Unfortunately, this would require a similar rewrite to the code as the second strategy.

While discussing how to increase performance, we considered whether we could round departure times and flight times to the nearest day (integer operations are generally slightly faster than floating point operations). Rather than requiring a difficult rewrite, this only involved changing the data types of the variables. Replacing the 8 byte double values we had been using with 2 byte shorts significantly reduced our output file size. With three such values for every transfer window (time of departure, duration of flight and transfer costs), this cut 18 bytes off for every transfer window in the database. As mentioned in section 2.1.1, NASA uses a resolution of 2-5 days, so our 1 day resolution is not unreasonable. Replacing the 4 byte transfer window counter with one of 1 byte improved the reduction. These changes reduced the file size by about a third.

C.3 The Improved File Format

The file format could be improved by making two changes. Although reasonably quick to implement, the risk of delaying the Map Generator's execution meant it was better to wait with the implementation. The produced file could instead be converted to the new format after completion.

The first change was to group entries originating from the same asteroid into one. Previously, all entries originating from the same asteroid were added sequentially to the file. The new file format changed this by removing all occurrences of the origin asteroid's name except for the first. Immediately after the name, a counter was added. The value of the counter is equal to the number of entries with the same origin asteroid. The new file format after the change:

```
entry = [origin name] [count c1 of destination asteroids]
        { [destination entry]^1, [destination entry]^2, ..., [destination entry]^c1}

destination entry = { [destination name] [count c2 of transfer windows]
        { [transfer window]^1, [transfer window]^2, ..., [transfer window]^c2 }
```

Additionally, asteroids in the database were assigned arbitrary 4 byte IDs to be used instead of the 26 byte names. An index in the beginning of the file, before all the entries, provides translation from asteroid name to the arbitrary ID. The IDs are arbitrary to the extent that they only apply to the file and the entries within it, but they are unique within the file. The format of the index is as follows:

```
index = [count i of index entries]
        { [asteroid name]¹[ID]¹, [asteroid name]²[ID]², ..., [asteroid name]ⁱ[ID]ⁱ }
```

The rest of the file is modified to use IDs rather than names:

```
entry = [origin ID] [count c1 of destination asteroids]
        { [destination entry]¹, [destination entry]², ..., [destination entry]^c1}

destination entry = { [destination ID] [count c2 of transfer windows]
        { [transfer window]¹, [transfer window]², ..., [transfer window]^c2 }
```

C.4 File Format Efficiency Improvement

The reduction in the size of a test file can be seen in Table 23. The table shows actual file sizes for a tiny problem size; less than 0.05^2. The 1200m/s delta v limit put on the transfer cost means that some asteroids are within the range of the BEEs but not counted as reachable due to a large transfer cost. Increasing the density of asteroids in the model allows the more "distant" asteroids to be reached by providing intermediate asteroids that can split the total transfer cost over several transfers. Consequently, the number of asteroids that are reachable grows faster than the density, so a density of 0.05 produces an even lower problem size than 0.05^2. It is worth noting that the asteroid database was filtered to 3400m/s, not the 1700m/s used for the full Transfer Map.

File Size Improvements

		File Size (KB)	*% of original*
Original		5,514.2	100%
Grouped Entries Only		3,280.7	59.5%
Final Format	Data	1,249.8	22.7%
	Index	646.5	11.7%
	Combined	1,896.3	34.4%

Table 23: File sizes for a problem size of 5% (22067 asteroids, 3048 reachable), database filtered down to 3400m/s, a BEE delta v limit of 3400m/s and transfer costs limited to 1200m/s.

Looking at the table, the original file represents the file produced by the full run using the old format, only at less than a 0.05^2 of the size. Grouping entries with the same origin reduced the file by roughly a third, not a quarter as previously estimated. Because the original format already incorporated the data type savings described previously, the larger reduction can be explained as the same percentage reduction applied to a larger percentage of the total data.

The final format improves upon the grouped entries format by also including an index of all asteroids in the asteroid database. The experiment shows that the final format is about three times more efficient than the original, and roughly twice as efficient as the grouped entry format. Three numbers are provided in the table: the size taken up by the transfer data, the size required by the index, and their combined size. The reason for this split is that the two components grow at different rates. While the index grows linearly in the number of asteroids in the database, the growth of the transfer data is $O(reachable^2)$. The experiment showed that about a third of the file size was used by the index. As the problem size increases, this percentage should decrease and the relative efficiency of the final format improve towards a file size a fifth of the original.

Appendix D Transfer Map Memory Usage

It turned out that the in-memory representation of the Transfer Map had an overhead of several hundred percent. This appendix describes the issues faced in reducing memory usage and how they were overcome, with the end result that the larger Transfer Maps would barely fit in system memory.

D.1 Investigating the Initial Memory Usage

At one stage, a large Transfer Map needed to be converted from the initial file format to the improved file format (both described in Appendix C). Unfortunately, this involved loading 61,581,451 transfer windows from the file to a data structure in memory. The code functioned as intended, though the process was killed by the operating system halfway through the file due to excessive memory use. The structure in memory was apparently even less efficient than the initial file format, requiring upwards of 10GB to load the 2GB initial file. If the search continued for much longer, we would not be able to resume the search or process the resume file at all.

Although the resume file could be finalised by converting it entry by entry without loading the entire file to memory at once, this would not allow the search to be resumed. We instead chose to improve the efficiency of the memory structure. The initial structure grouped all entries for a pair of asteroids together in a list and packed these into a two dimensional index allowing for quick access.

```
HashMap of Origin and (HashMap of Destination and List of Transfer)
```

Table 24 shows lower bounds on the memory requirements for the different components of the data structure. Actual usage depends on factors like memory alignment (padding), requirements specific for the implementation and behind the scenes book keeping (for instance heap allocations). An example is the transfers, which require 6 bytes of storage. On an x86 or x86_64 architecture machine, they are likely padded by 2 additional bytes to place each Transfer neatly on the beginning of a memory word, every 4 or 8 bytes. The table does not include this extra overhead for simplicity, but still shows where reductions are likely to help.

Memory Usage of Original Transfer Map Structure

	Size (Bytes)	*Approximate Number*	*Estimated Usage (MB)*
Transfer	6	60,000,000	330
Transfer*	8	60,000,000	440
std::vector	24	30,000,000	690
std::string	59/8	30,000,000	1,700/240
std::unordered_map	12,000	65,000	740
Total	-	-	3,900/2,440

Table 24: Shows the various contributors to excessive memory usage in the original resume functionality. The memory required by the strings depend on whether strings are duplicated in memory or not.

D.2 Modifying the Memory Structure

Changing the three 8 byte double precision numbers in our transfers to three 2 byte shorts reduced their size to less than that of a pointer, which is 8 bytes on a 64bit system. The pointers therefore no longer served a purpose and were removed relatively quickly. The pointers were used to ensure persistence of the transfers within the application, but were no longer needed as the transfers would always belong to a single unique object.

The transfer windows of an asteroid pair were grouped in an std::vector, a small wrapper around a

bare (C-style) array. There were 64,552 origin asteroids, each with approximately 500 possible destinations. Although the 24 bytes are usually insignificant, with close to 30 million of them they do add up. A more lightweight solution was to store every transfer window in a single list, and instead point to the range of transfers belonging to the asteroid pair. This would eliminate all but one list and introduce two numbers; the first and last index of the range.

There were two choices of pairs; the indices could either be represented by two size_t (aka unsigned long integer) or two plain unsigned integers. Both would be of the type std::pair and have zero overhead. The size_t is guaranteed to always be able to represent the largest index the system is capable of handling and not be any larger, which means 8 byte on our 64bit machines. However, plain (possibly) 4 byte integers would be able to handle indices up to roughly 2^{32} (~4,300,000,000), a few orders of magnitude more than we have memory to handle. But an unsigned integer could also represent a 2 byte integer, which would only support up to 2^{16} transfer windows (~65,000). As integers are 4 bytes on all our machines, we deemed the reduced cross platform support worth the 50% reduction in memory consumption.

The effective size of std::string in our application is not easy to estimate. For the trivial case of having only one occurrence of one name, the overhead is tremendous. The 26 character string must be appended with a null character ('\0') and stored in a character array on the heap. This requires 27 bytes for the array and an 8 byte pointer to it. The string class stores the pointer to the array, an 8 byte number for the length and another 8 byte number for the capacity, as well as another 8 byte number to count the number of references to the string. Finally, an 8 byte pointer to the string structure is needed for a total of 59 bytes.

Fortunately, the number is only valid if there are no duplicate strings, as our compiler will reuse strings if it can safely do so. The reference counters and extra pointers ensure that assignment only requires another pointer to the same string structure and the reference counter to be incremented. If the hash maps we use take advantage of this, they should only have to store an 8 byte pointer to existing strings, not the full 59 bytes. The table show the memory required for both cases.

The std::unordered_map is designed to store large numbers of elements and to store and retrieve them quickly. We required close to 65,000 of them, each with less than a thousand elements. As for the strings, the actual memory usage of one object is difficult to measure. The *C++* keyword sizeof() reported a class size of 56 bytes, not including the size of everything the class variables reference through pointers.

The hash map keeps track of buckets representing one hash value, containing a list where all elements of that value are stored. For efficient lookup, the map attempts to keep the load factor close to 1, meaning on average one bucket for each element. Although the limit can (and possibly should) be overridden, it still means an additional list for each bucket. In our case, the total number of buckets and lists should be close to 30 million. A small overhead quickly adds up when repeated millions of times.

As can be seen, Table 25 accounts for less than half of the actual memory usage. The numbers found in the table do not consider memory used behind the scenes, for instance memory padding and the two 8 byte memory words required for every heap allocation. The estimates are also lower bound, as not all memory usage of the effectively black box library classes can be discovered by casual investigation.

No suitable way was found for eliminating the index, so the overhead of the hash maps themselves was not resolved. However, one of their main sources of overhead was the choice of asteroid names as keys. The only consistent identifier for the asteroids was their names, making it the obvious choice. Part of the change to the improved file format was creating an index translating from asteroid names to asteroid ID, valid only within the contents of the file.

Memory Usage of Improved Transfer Map Structure

	Size (bytes)	*Approximate number*	*Estimated usage (MB)*
Transfer	6	60,000,000	330
std::pair<int,int>	8	60,000,000	440
std::string	59	200,000	11
int	4	30,000,000	110
std::unordered_map	12,000	65,000	740
Total	-	-	1,600

Table 25: Show lower bound on the estimated memory usage after changing the structure of the transfer map stored in memory.

By first passing the asteroid names through this index, the two dimensional transfer index could use the IDs as keys. The strings used as keys required at least 8 bytes, as discussed previously, while plain integers would halve this usage. The loading of initial file format was changed to translate the names from the file into IDs before adding them to the index. The loading of improved format was similarly simplified, as the name index of the file could be loaded and the transfer entries be applied directly to the index without any translation.

The changes took several hours to implement, partly due to experimentation with methods abandoned in favour of the ones discussed here. The apparent memory usage was eventually reduced down to less than 2GB with new estimates as seen in Table 25. Interestingly, although the actual memory usage was reduced by a factor larger than 5, the estimated memory usage was reduced by a factor of 2 only. Most of the savings came from sources we were not aware of or could not measure easily. The most likely choice was inefficiencies in handling the large number of strings. Another possibility was that hash maps could perform better when using integers rather than strings for keys.

D.3 Reducing the Time Needed to Resume

Changing the data structure of the transfers in memory warranted a test of the resume functionality to ensure that it still worked. Resuming was unfortunately painfully slow. Resuming had previously only required skipping calculation for a few thousand asteroids, but now had to skip over 60,000. Our development computer managed to skip roughly 500 every minute, which meant a full resume would take a couple of hours. Additionally, as the resume process did not take advantage of parallel computation, our main processing computer would only manage slightly more than that.

A couple hours was spent trying to increase the single-thread performance of the resume functionality, but only about 20% of the run time could be eliminated. Most was spent performing lookups in the index of transfer windows. This involved hashing the name of both asteroids in the pair to obtain their IDs (lookup in the name index) followed by hashing of both their IDs to obtain the indices into the transfer list (lookup in the transfer index). Performance was improved by not repeating the lookup procedure for the origin asteroid for all potential destination asteroids. It would be possible to iterate over the destination asteroids only and not every asteroid, but this would require a reverse index from asteroid ID to asteroid object. As there was an easier way of resuming quickly enough, this option was not pursued.

Multi-threading the resume loop of each asteroid was an easier option. Some changes were needed,

but the performance was improved two-fold on the dual core development computer, to roughly 1500 asteroid every minute. The processing computer was expected to see yet another doubling of performance due to its quad core CPU, improving performance by a factor of 5. This would allow processing to be resumed in less than an hour even for 100,000 asteroids.

Appendix E Handling the Transfer Map File

Once the Transfer Map Generator (described in section 5.1) had completed execution, the end result was a file containing every transfer window found during the search. These included those that were never actually traversed by the search, but kept in case later discoveries would allow their origin asteroids to be reached early and cheaply enough for traversal. This appendix describes the issue in more detail and the algorithms used to remove the untraversable transfer windows and unreached asteroids from the Transfer Map.

E.1 Finalising the Resume File

As can be seen in Appendix D, the memory footprint of the Transfer Map file was quite large. Additionally, the original 2.2GB file still required 500MB after converting it to the new file format. Pruning the untraversable transfer windows would reduce both the amount of memory required and the size of the file for little extra development time.

The untraversable transfer windows had one thing in common; they could never be part of any valid route because the origin asteroid could not be reached in time for the departure with enough remaining delta v to complete the transfer. All such windows could be removed by an exhaustive search from the HIVE, enumerating all valid routes through the asteroids belt and keeping only the windows used during the search. After which any unreachable asteroids could be removed as well. Such a search would have a high computational complexity, requiring every possible route to be examined. A conceptually more complex, but computationally simpler algorithm would eliminate most of the junk data.

Junk data comes from a couple of situations: either the transfer window cannot be reached with sufficient propellant to make the transfer or it cannot be reached in time to make the transfer. A third option is that the transfer window can be reached with sufficient propellant, but only after the window had closed, while all earlier routes are too costly.

The first group make up the leaves of our search tree, included in the file because we could not rule out that a later, cheaper route would be found. We can find the cheapest way to reach a given asteroid (ignoring the issue of time) by using the same methods used in the search itself. As seen in Algorithm 8, this allows us to remove any asteroid which has a cheapest access route which is beyond the BEEs' capacity, and any transfers to or from such asteroids. In addition, any transfers out from an asteroid which would require more propellant than what the BEE can have left at that point are removed.

```
For every pair of asteroids (asteroid1, asteroid2):
     Store the cheapest transfer window from asteroid1 to asteroid2

Initialise every asteroid's cheapest cost to be infinitely large
Initialise the HIVE's cost to 0
Put the HIVE in the queue

While the queue is not empty:
     Retrieve asteroid from queue
     For all cheapest transfer windows originating from the asteroid:
          If origin's cost + transfer window's cost < destination's cost:
               Update destination's cost
               Put destination in queue

// All asteroids now have their cheapest cost set correctly assuming no time issues
For all transfer windows:
     Remove if cost of transfer + cost of origin exceeds the delta v limit
Remove any asteroid whose cheapest cost of arrival exceeds the delta v limit
```

Algorithm 8: Pseudocode method for pruning away unreachable transfers based on a delta v limit.

A similar method is used to remove windows which cannot be reached in time. Instead of keeping track of the most fuel efficient way to reach each asteroid we keep track of the earliest available transfer to it. After finding the earliest possible arrival time for each asteroid (ignoring the issue of available delta v), any windows which would require leaving before this can be removed from the Transfer Map. Any asteroids which cannot be reached with the transfers in the file can also be removed, along with all transfers to and from such asteroids.

Pseudocode for eliminating intractable time requirements can be seen in Algorithm 9 below. The algorithm begins at the HIVE and estimates the earliest arrival for all asteroids reachable from it. Whenever an asteroid's estimate is updated, it is placed at the back of a queue for future processing in case it can provide a better estimate to other asteroids. At the end, both transfer windows that could never be used and asteroids that could never be reached are removed.

```
Initialise every asteroid's earliest arrival time to the end of time
Initialise the HIVE's arrival time to 0
Put the HIVE in the queue

While the queue is not empty:
     Retrieve first asteroid from queue
     For every transfer window originating from it:
          If time of departure after the origin's earliest arrival
               And the arrival is before destination's earliest arrival:
                    Update destination's earliest arrival to window's time of arrival
                    Put the destination asteroid in the queue

// Any asteroid now has its earliest arrival time set to the earliest possible.
For all asteroids:
     If earliest arrival is after mission end:
          Remove the asteroid and all transfer windows originating from it
     Otherwise:
          Remove transfers with departure time before the asteroid's earliest arrival
```

Algorithm 9: Pseudocode method for removing intractable time requirements.

Finding the last group, which cannot be reached cheaply enough in time for the transfer, would require searching every possible route in the transfer map. This would require several orders of magnitude more computing power than searching through the cheapest or earliest transfers, and significantly more memory too. Given the enormous resources required and the relative little gain expected, we decided to let this particular source of junk data be, until and unless we find that there is still too much remaining.

We expected the time restrictions to remove more windows than the delta v restrictions, as the delta v restrictions were already to some extent coded into the search itself and therefore could only remove leaf data. The time of departure restrictions, on the other hand, could render asteroids

inaccessible near the HIVE, removing the asteroid and its entire sub tree in one strike. By first eliminating time restricted windows, fewer untraversable windows would be left when eliminating leaf data, speeding up the process.

E.1.1 Pruning a Test File

A test file was pruned to test the effectiveness of the pruning method. The results are shown in Table 26. The original file was produced from a run with the same settings as for the previous file format experiment (section C.3). One can see that time pruning removes a larger portion of the file. Although delta v pruning alone removes more than half of it, most of that is also removed by time pruning. This is to be expected as only leaf data is removed by the delta v pruning, which is also the data most likely to break the time constraints.

Finalisation Performance on a Test File

	In Original	*Removed by Time Pruning*	*Removed by Delta V Pruning*	*Removed by Both Combined*	*Remaining After Pruning*
File size	1,896 KB	-	-	1,788 KB	108 KB
	100%	-	-	94.3%	5.7%
Asteroids	7,078	5,826	4,030	5,842	1,236
	100%	82.3%	56.9%	82.5%	17.4%
Transfer windows	132,993	125,170	91,736	125,257	7736
	100%	94.1%	69.0%	94.2%	5.8%
Index size	22,067	-	-	20,831	1,236
	100%	-	-	94.5%	5.6%

Table 26: Shows the effect of pruning a resume file from a problem size of 5% (22,067 asteroids, 3,048 reachable), database filtered down to 3,400m/s, a BEE delta v limit of 3,400m/s and transfer cost limited to 1,200m/s.

Table 26 shows that approximately 30% of transfer windows would remain after delta v pruning and roughly 5% after time pruning. Two things were obvious: significant time was spent calculating transfer windows that could never be used and far fewer would be calculated if time constraints were enforced by the search. However, actually estimating the speedup without changing the search algorithm would be difficult and mostly rely on guesswork.

Assuming that a search restricting both on time and delta v would produce the same proportion of leaf data as a search restricting on delta v only, it should also produce 70% junk transfers and 30% useful data. We see that only 5% of calculated data remains after pruning for both. In addition to that useful data, we can expect two and a third (70%/30%) times as much junk data. In total, we would then end up needing to calculate three and a third times the 5% we actually want, which is 16.67% or a 6^{th} of the current total.

Another way of looking at it is to consider the number of useful asteroids with and without time pruning. Delta v pruning keeps 3,048 asteroids, the exact number of origin asteroids for which transfer windows have been calculated. Time pruning reduced this to 1,236, roughly 40%. In other words, if this held true for the search, we would see a 60% reduction in run time.

Regardless the actual speedup, it would seem there was a significant improvement to be made by incorporating time restrictions in the search algorithm. It would be most appreciated even if it were only 10-20%. So why did we not initially do this? In hindsight, it is difficult to fathom, as it should

have been obvious at the time.

When designing the search, we first considered limiting each asteroid's search area to after the asteroid's estimated earliest arrival. This would be very efficient in terms of the amount of the pork-chop plot searched, but would be a lot more complex to code. Whenever a new and earlier route to an asteroid was found, a new slice of the pork-chop plot would have to be searched and duplicate transfers removed.

The Transfer Map generator was only supposed to be a small side project, but quickly turned out to require far more effort than originally thought. Writing a complex generator of transfer maps in the asteroid belt was never the goal. As we abandoned the search described above for a simpler search of the entire pork-chop at once, time restrictions were forgotten. Not before we devised the time pruning did we consider it again. Although tacking on an extra restriction on time for the same search would require relatively little extra development time, this strategy did not occur to us until the initial search had already started. It was, naturally, included in the second search (section 5.7.1).

E.1.2 Pruning the Transfer Map

It was, unfortunately, not possible to complete the final search given our strict deadline. The resume file was pulled from the computer and the searched allowed to progress in case a larger map would be needed at a later stage. The resume file was finalised as described in section E.1, with results shown in Table 27.

Finalisation Performance on the Transfer Map

	Original	*Remaining after Time Pruning*	*Finalised*
File Size	1,140 MB	-	464 MB
	100%	-	40.7%
Transfer Windows	129,426,200	48,696,908	48,696,755
	100%	37.6%	37.6%
Asteroids	208,386	159,499	159,499
	100%	76.5%	76.5%

Table 27: From left to right, the columns show three attributes for: the original file, file after time pruning, and the finalised file after pruning for both time and delta v.

The search visited 42,128 asteroids in the 20 days it was allowed to run, far from the worst case of the roughly 210,000 asteroids found in the database. How close the search was to completion can only be guessed. In terms of data, the raw resume file grew to over a gigabyte. This is roughly twice the size of the 500 megabyte resume file generated by the previous search before it was stopped in favour of this search. What is interesting is that the previous search visited more than twice the number of asteroids but produced half the number of transfer windows. This search apparently produced four times the number of transfer windows for every asteroid on average.

The table clearly shows how limited the pruning algorithm is. The search only visited roughly 42,000 asteroids, while the pruning considered almost 160,000 asteroids reachable. This means a whopping three quarters of data not removed by the pruning should be considered junk. Section E.1 describes the pruning algorithm and why some sources of junk data were left alone due to time restrictions. We knew that some of it would in reality be junk data, but did not expect this to be the majority.

Table 26 shows that the pruning algorithm removed roughly 94% of the data from the previous search algorithm. This led us to believe that a speedup of up to 6 could be achievable by changing it. The new data from Table 27 showed that the speedup would probably be higher than this.

Assuming that three quarters of the data remaining after the pruning was in fact junk, the estimates should be multiplied by four. This changes the estimated speedup of 6 to 24, and the estimated 60% reduction in run time to an estimated 90% reduction. As stated previously, the numbers were not reliable, and the updated ones may not be much better.

E.1.3 Improving Pruning Algorithm

The pruning algorithm described in section E.1 was extended by including a third pruning condition before the two others. The original two conditions first removed any asteroid and window that could not be reached when observing restriction on time, effectively disallowing time travel. The second condition removed asteroids and windows that could not be reached within delta v constraints. They did not, however, remove asteroids that could never be reached and windows that could never be traversed by any valid route.

The new search allowed a third condition to easily be put in place: removing non-visited asteroids. Any asteroid not yet visited by the search would be an asteroid not presently reachable by any valid route. They are only present in the Transfer Map file as destinations, and only because they could end up being reachable after the discovery of new reachable asteroids.

Although this condition would also have worked with the previous search algorithm, it would not have worked nearly as well as with the new search algorithm. The previous visited far more asteroids than were actually reachable by a spacecraft, and any such asteroid would not be removed. With the new search, these asteroids would not have been visited, and would therefore be removed by the additional condition.

```
For all <origin, destination> pairs in index:
    If there does not exist a pair in index with destination at the first position,
        remove the pair from the index
```

Algorithm 10: Pseudocode for pruning non origin asteroids.

Removing non-origin asteroids is very simple. The algorithm, outlined in Algorithm 10, simply goes through the index and removes any destination asteroids that are not also present as an origin asteroid. The transfers belonging to the removed asteroid pairs are simply left in the transfer list to be discarded once the program ends. They are not copied to the finalised Transfer Map because they are not referenced by the index.

Finalisation Performance on the Transfer Map with Improved Algorithm

	Original	*Remaining after Pruning Non-Origins*	*Remaining after Time Pruning*	*Finalised*
File Size	1,140 MB	-	-	324.5 MB
	100%	-	-	28.5%
Transfer Windows	129,426,200	73,292,203	34,532,069	34,532,005
	100%	56.6%	26.7%	26.7%
Asteroids	208,386	42,128	42,128	42,128
	100%	20.2%	20.2%	20.2%

Table 28: Pruning the Transfer Map with the improved pruning algorithm.

As can be seen in Table 28, the additional condition improved the effectiveness of the pruning

algorithm significantly. It was particularly the reduction in the number of asteroids that was important, as each pair required an additional entry in the index. The overhead for storing this in memory was quite significant, as described in Appendix D, mostly due to the way hash maps function. Every asteroid pair that was pruned meant the removal of one hash, one key, and one list from the index. Transfer windows that were removed with the pair were an additional source of improvement.

There should still be some junk data left, as the pruning algorithm had not been changed to take actual routes into account. It worked on the assumption that the earliest route was the cheapest route and vice versa. Performing a proper search with matching transfer cost and time of departure/arrival would require a radically different approach.

E.1.4 Reducing the Memory Requirements of the Pruning Algorithm

We were unfortunately not able to finalise the Transfer Map file straight away. The file was twice the size of the one from the previous search, which required optimisation as described in Appendix D. The problem we faced with the previous resume file was that the memory overhead of storing the transfer windows and their metadata in memory was too large.

The problem this time around, however, was the pruning algorithm, which allocated extra data to store the pruned transfer map before removing the original from memory. This was not an issue with the previous search, where close to 95% of the data was not duplicated. Not only did this search produce a file twice the size of the previous, it retained far more of the data than the previous. The effect was that loading the file took most of system memory, and pruning would have exceeded it.

The solution was to not duplicate any data, even for a short amount of time. Previously, a new list of transfer windows was created containing only the ones to keep. A new index of asteroid pairs and indices was made containing only the pairs with windows left. Instead of this, the new pruning algorithm reordered the transfer list, inserted asteroid pairs with windows left into a new index and removed the asteroid pair from the old index. The result was a transfer list containing sequences of valid transfer windows separated by sequences of undefined data, and an asteroid pair index pointing only to the valid sequences.

Reordering the transfer windows was an uncomplicated task. A range of transfer windows was associated with each pair, with begin and end indices stored in the index. Beginning at the start of the index, each window was inspected to see if it should be included in the finalised file. Each time a window was found that should not be included, the remaining windows were moved towards the beginning as if the original window had been removed. The end result was a smaller range with the same begin and a smaller end.

If there was at least one transfer window left in the range, its indices were inserted into a new asteroid pair index. The asteroid pair was then removed from the old index. As the search progressed, the result was a slight reduction in memory usage rather than a large increase.

E.2 Loading the Transfer Map Into the AI Simulator

Loading the transfer map file into the simulator should have been a straightforward process, and for the most part, it was. There was a problem, however: Java read the bytes in the wrong order. At first, we suspected there was an issue with the file or the logic for reading it. Inspecting the beginning of the file using a utility called “hexdump”[43] revealed that the contents were correct, however. The Map Generator, written in *C++*, simply wrote the number directly to file, byte for byte from memory. The AI Simulator, written in Java, was supposed to do the opposite. Although it read the correct bytes, the Simulator produced completely different numbers.

It turned out this was due to a difference in endianness in *C++* and Java. Machines using the x86 architecture store numbers in memory in Big-endian format. This applies to any code running on the machine, *C++* or Java alike. This meant that the *C++* code wrote Big-endian numbers to file. Unfortunately, Java's file reading functionality assumed the file was written in Little-endian with no accessible way of reading Big-endian.

For instance, the first number of the file was the file version number, 1, used to identify the file format. On a Big-endian machine, this 4 byte integer is represented as `<0x01,0x00,0x00,0x00>`, which was what the file contained. Java's methods read the integer as if it were Little-endian and converted it to Big-endian, producing the number `<0x00,0x00,0x00,0x01>` in memory.

If it were *C++*, this could easily be solved by manually reading in 4 bytes, swapping the order of the bytes and interpreting them as an integer. This is actively discouraged in Java, however, and required the use of a *ByteBuffer* wrapped around the stream. The buffer required a special flag to convert the bytes from “Little-endian” to “Big-endian” before Java could correctly interpret the bytes. After a number had been read, the buffer had to be reset and the process repeated.

After verifying that the numbers were indeed interpreted correctly, another issue became apparent: the strings of the file seemed unintelligible and the file failed to load correctly. It seemed Java assumed strings were written in UTF-16 (two bytes per character) with no way of overriding the methods to read the (one byte) UTF-8 characters produced by *C++*. Strings had to be manually read as bytes, then interpreted as a series of characters and finally converted to text strings.

E.2.1 Improving AI Simulator Transfer Map Loading

When loading the pruned and finalised transfer map into the AI Simulator, we faced a daunting problem. Up to this point, we had only used transfer maps with a size of a few megabytes after finalising. This Transfer Map required roughly 400 megabytes. It was quickly discovered that the same issues that plagued the *C++* implementation also applied to the AI Simulator; the memory overhead was too large to load the file.

The first issue was that one of our development computers had 32bit Java installed, which apparently only supported a heap size of up 1.5GB, far from what we needed. Removing the 32bit Java and 32bit Eclipse IDE used for development and installing the 64bit variants solved this.

Once the loading could progress to fill all available system memory, another problem was revealed. Reading the Transfer Map, a binary file, took far longer than expected. Loading the smaller transfer maps we had previously used did not take long enough to notice the problem. It turned out that Java's file reading and input streams do not buffer reads and writes; a dedicated buffer stream seems to be required. Coming from *C++* where buffering is not only readily available but enabled by default, this came as a surprise. Adding a *BufferedInputStream* between the *FileInputStream* (file reading) and the *DataInputStream* (byte reading) solved the problem. The time required to load the entire file was reduced from an estimated 45 minutes to roughly 5 minutes.

Loading of the file still only progressed to roughly 30,000 out of close to 130,000 asteroids on one development computer before stalling. At this point, the Simulator stopped at a memory usage of close to 4.0GB and started using all cores fully although the loading was performed sequentially. This suggested some sort of internal garbage collection thrashing, which was found to be the most probable cause.

The fact that loading the entire file would likely require 17GB was a source of concern. This was far more than could be made available. To reduce the memory requirement of loading the file, the structure of the map in memory was changed. The original used the same structure as originally used in the Generator: two dimensional hash map with strings as keys and a list as the final value. The list was swapped out for a Java array, which has a lower overhead than the list. Loading

stopped at roughly 40,000 asteroids after the change, suggesting a new total estimate of 13GB.

This was still more than we could handle, and it was decided that an improved pruning algorithm would be the best way to solve the problem. The process of improving the pruning is described in E.1.3. The changed algorithm removed 25% of the file size and reduced the number of asteroids to the number of asteroids visited. At this point, the Transfer Map stored in memory required roughly 2.6GB. The main problem was the second and third steps, however: loading the asteroid values and classes and combining them with the transfer map for a complete representation usable by the Simulator.

Unfortunately, the internal representation used by the Simulator required more memory than the optimised representation used by the Map Generator. To make matters worse, the code as written created the internal representation before allowing garbage collection to collect the data found in the transfer map. This effectively duplicated the contents in memory until loading had completed, which meant it could never complete. It would be best to reuse the technique of removing the data immediately once it was no longer needed. This slowed down the memory growth by releasing the roughly 2.6GB during the third step of loading.

Appendix F Improving the Search Algorithm

Pruning untraversable transfer windows from the Transfer Map (see Appendix E) revealed a source of improvement in the Generator's search algorithm. Inspection of initial results suggested the search would benefit from allowing the duration of flight to be limited. While sections 5.7.1 and 5.7.2 describe the changes made to accommodate these modifications, this appendix covers the performance measurements to analyse their effects.

F.1 Benchmarking the Improved Search Algorithm

Table 29 shows the run times and asteroids visited for increasing problems sizes. The table only covers a fraction of the total problem size, but still offered some insight regarding the total run time requirements. Due to the high run time of the old algorithm, some of the values are missing. It required roughly 18 hours to generate a map with a density of 8%, suggesting that the final two values would likely require upwards of 24 hours each.

Comparing the old and Improved Search Algorithm

Problem Size		*Search using cheapest transfer (old)*		*Search using cheapest and earliest transfer*		*Search using valid routes (new)*	
Density	*Asteroids*	*Visited*	*Time (min)*	*Visited*	*Time (min)*	*Visited*	*Time (min)*
0.01	2,096	95	2.29	14	0.37	14	0.34
0.02	4,142	554	26.5	39	2.01	39	1.89
0.03	6,194	1,274	95.3	80	6.21	80	6.16
0.04	8,270	1,997	201	128	13.2	128	12.6
0.05	10,415	2,850	365	180	23.0	178	22.5
0.06	12,551	3,602	551	251	38.3	249	36.2
0.07	14,727	4,424	797	331	59.4	329	56.3
0.08	16,846	5,282	1,076	398	83.0	396	76.7
0.09	19,021	-	-	472	104	470	102
0.10	21,155	-	-	586	143	584	141

Table 29: Shows the run time (development computer) and number of asteroids visited by three variants of the search algorithm. Leftmost is the old search algorithm, which allowed branching based on cheapest transfers only. Part of its results are mission due to excessive run time requirements. The middle shows an intermediate algorithm adding restrictions on time, allowing branching using the cheapest and earliest arrival without considering routes. The rightmost algorithm is the new search which requires valid routes, considering both time and delta v.

The number of asteroids visited is a more reliable metric than the run time. The run time can be influenced in a number of ways, such as the activity of other processes and changing air temperature (cooling). The measurements were also made on a development computer with approximately half the computing power of the computer designated for map generation.

Looking at the results, one can see that the additional restriction on time reduced the number of asteroids visited to a fraction. The original search would have completed in a fraction of the time had we made the changes before starting the search, even if development would have delayed start up a day or two.

On the other hand, it would seem we were not wrong when we chose to use estimates rather than actual routes. Only two asteroids were included in the estimate-driven search when they should not have been. The route-driven variant is slightly better, however, and there is no reason not to chose it. Run times even show that it is slightly quicker. This may be an artefact of the experiment, which did not control for tasks running in the background or room temperature and the likes.

The intermediate and new algorithms performed similarly to one another and both vastly outperformed the old search algorithm. A major speedup was expected, maybe even by a factor of 6 (section E.1.1). Table 30 shows this estimate was pessimistic. The data points suggest a speedup of roughly 13.7 (the average) and twice what was estimated.

Speedup of the Improved Search Algorithm

Density	*Asteroids*	*Speedup*
0.01	2,096	6.79
0.02	4,142	14.2
0.03	6,194	15.9
0.04	8,270	15.6
0.05	10,415	16.0
0.06	12,551	14.5
0.07	14,727	13.4
0.08	16,846	13.3

Table 30: Shows measured speedup when using the new search algorithm, data taken from Table 29. The speedup is here defined by the reduction in asteroids visited.

F.2 Benchmarking the Modified Search Space

An estimate of the run time requirements was needed before the second search, described in section 5.7, could be started. Table 31 shows the complexity and run time of several small problem sizes.

Extended Mission Run Time Measurements

Problem Size		*Solution*		
Density	*Asteroids in Search*	*Asteroid Visited*	*Percentage Visited*	*Run Time (hours)*
0.01	2,096	417	19.9%	0.08
0.02	4,142	1,680	40.6%	0.66
0.03	6,194	3,298	53.2%	1.95
0.04	8,270	4,921	59.5%	3.84
0.05	10,415	6,928	66.5%	6.46
0.06	12,551	8,733	69.6%	10.4

Table 31: Number of asteroids visited and run time required on development computer for various small problem sizes. Search was performed with a Nelder-Mead grid resolution of 15 and maximum duration of flight of 4 years.

The percentage of asteroids that are reachable by the search increases towards 70%. The most probable cause for this is the 1200m/s limit put on transfers. As more asteroids are available, the likelihood of finding a valid route to an asteroid increases. The subset of asteroids is also chosen at random, making it representative of the full distribution except for the relative distances between asteroids. It is therefore natural to assume the percentage will at least stay at 70% and probably increase beyond what has been seen. Because the search ended at 6% density, it is assumed the search will visit a larger percentage of asteroids, for instance 80%.

As described in Appendix B, the run time can be described as the product of three factors: the

number of asteroids in the search, the number of asteroids visited by the search and a constant representing the time needed to find the transfers from one asteroid to another. In this case, the first is known to be 212,077 and the second estimated at 80% of this number, 169,662 asteroids.

The estimated run time of a full search on the development computer can then be computed by scaling the results of the largest search up to the full density. The result must then be adjusted according to the speed difference between the search and development computers. The computation is shown below:

$$run\_time_{estimated} = \frac{full_{asteroids} \cdot full_{reachable}}{reduced_{asteroids} \cdot reduced_{reachable}} \cdot reduced_{run\_time} \cdot speed_{development}$$

$$run\_time_{estimated} \approx \frac{212{,}077 \cdot 169{,}662}{12{,}551 \cdot 8{,}733} \cdot 10.4\,hours \cdot 0.4 \approx 57\,days$$

Roughly two months of computation was unfortunately well outside what could be achieved; two to three weeks at most. There was little that could be done to reduce this further except for acquiring a more powerful computer. The upside was that the new search algorithm should produce roughly an order of magnitude less junk data than the previous search algorithm, which ran for almost 50 days. Running the new search for no more than a week would still likely produce a transfer map the size of the one found by the first search. There was therefore not be a lack of data despite the search not being allowed to run for more than 20 days.

Appendix G Minor AI Simulator Components

This appendix covers minor components of the Artificial Intelligence Simulator that were either abandoned or not important enough to cover earlier.

G.1 Optimise Genome

One of the more unusual operators we chose to implement was a genome optimiser. This new step would look for alternative transfer windows available to the BEE for a given origin and destination. If a cheaper window could be found which did not arrive any later than its current window, there was no reason not to take the alternative transfer instead.

The goal was to increase the amount of time and delta v available to the spacecraft after any given transfer without damaging the flexibility of the evolutionary algorithm. Hence, we had to limit ourselves to unambiguously better transfers; any trade off between arrival date and propellant use would have to be up to the AI.

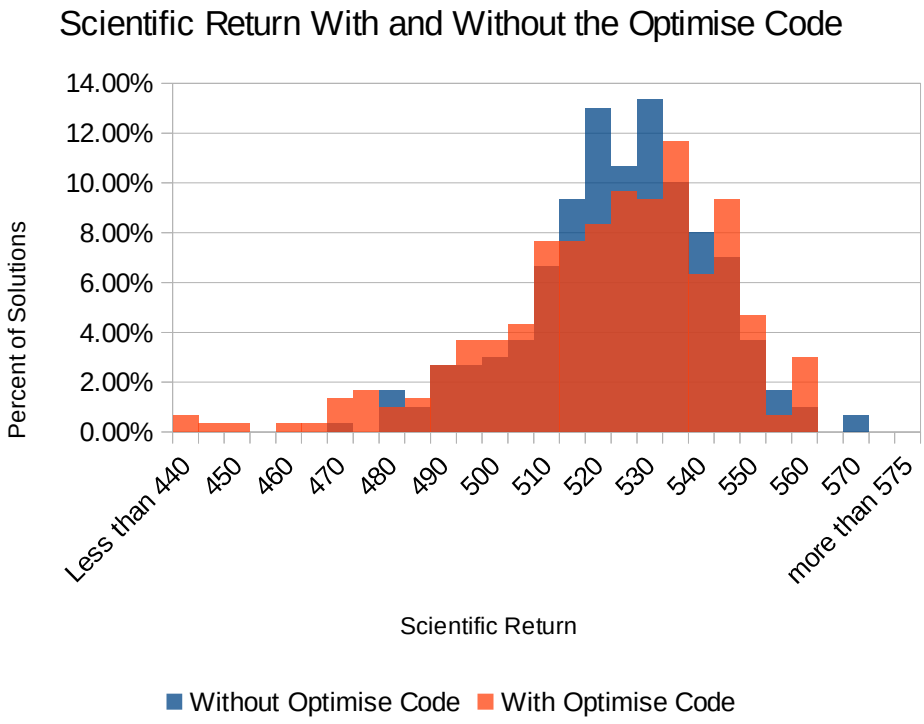

Figure 40: A comparison of the scientific return from a No Communication experiment with and without the optimisation code.

However, despite our best efforts, the optimisation code had very little effect on the utility of no communication runs. It did increase the average number of asteroids visited by about 0.5 asteroids, but at the same time it reduced the average scientific return by 1%. The mean scientific return was unaffected by the code, though the poor solutions were significantly worse with the optimisation code than without it.

G.2 Alternative Genome Encoding

As our crossovers were not really having the effect we would have liked, it was worth while to consider some alternative ways to encode the genomes. Alternative direct representations would have the same issues as our current encoding, due to the sheer width of the search-space. With many

thousands of possible asteroids, there is little chance of two genomes visiting the same one.

So we would need to use an indirect encoding. A simple possibility has been described in section 4.5.1. It would involve having each gene be a number of transfers to ignore before taking the next available transfer. While this is not a particularly good solution, a few minor improvements could make it at least workable.

The first simple improvement would be to include a target asteroid class in the genome. Instead of taking the n^{th} possible transfer out from its current asteroid, it would take the n^{th} possible transfer out to an asteroid of the chosen class. Such an encoding would have to handle the situation when there are no transfers available to that particular class, however. We could simply have a list of classes sorted in order of rarity, and if no transfers to a given class is available the transformation function would simply try the next class in the list. Given that the vast majority of asteroids belong to one of three classes, it may actually be better to have four different options here, with the last simply being “other”. This also has the useful feature of being encodable in two bits.

Another possibility would be to target asteroids of a given value range. The value of a given asteroid is a double value ranging from 1.0-10.0 (exclusive), so each gene could for example have an integer from 1 to 9 and only target asteroids with a value in the range of that number to one higher. Again, it would have to handle the situation of there not being a possible transfer window with that value, probably by expanding the search each to both higher and lower values until transfer windows are located.

The benefit of these changes is that they would allow the BEEs to retain specialisations without requiring them to target specific asteroids. Specialisations on class, departure time, and value are all possible. The downside, as described in section 4.5.1, is that they cannot reliably specialise on specific routes or sections of the search space.

A simplified version of this design was coded up and integrated with the rest of the simulator. The major simplification was that if it failed to find a transfer which fit its value and class criteria, it would revert to taking the n^{th} possible transfer overall. Similarly, the n was modulo the number of possible transfers at the time.

Unfortunately, the results from this implementation were discouraging. On average, the scientific return was slightly below that of the original genome, but that was not a major concern. More troubling was that the solutions had the same issues as the original genome. They still ran out of time more often than propellant. This suggested that our transfer map was the real limiter for further improvement, not the genome encoding.

One interesting thing to note is that, since one in four genes initially targets classes other than the three assigned classes (see section 5.3), the alternative genome actually was better exploring a variety of classes. Not by much, since very few asteroids in our data-set have known classes, so virtually all have gotten one of three classes assigned. But enough to be noticeably different to the original genome encoding.

G.3 Graphviz

The main thing we wanted out from the AI simulator was a list of scientific returns from each trial run. From these numbers, we would be able to run statistical analysis and discern how much of an effect communication had had on the amount of exploration done.

In order to ensure that the scientific return was a useful metric, we also printed out how many unique asteroids had been explored in each simulation. This made it easy to see whether higher scientific return corresponded with more exploration. In addition, it also gave more of a view into what was going on inside the simulation. But it was not sufficient to let us debug the process.

We wanted to be able to look at the routes used in the simulation in a way which made it easy to see where the BEEs had travelled. We could easily print out the routes themselves, but that would make it difficult to see when multiple BEEs had visited the same location. While some asteroids have names like “4942 Munroe”, which are relatively easy to distinguish from other asteroids, most have less readable names like “241530 2010 CC77”.

Our solution was to print the routes in a format readable by Graphviz[44], an open source graph visualisation software package. Its syntax is fairly simple. Nodes are represented by text strings surrounded by quotation marks. Edges are represented by an arrow made up of a dash and a greater-than sign (`->`). Square brackets can be used to assign properties to both nodes and edges. The syntax used for describing a single transfer is shown below in Algorithm 11.

```
"Origin_Asteroid_Name (Origin_Class)" -> "Destination_Asteroid_Name
    (Destination_Class)" [ label = "dv: Delta_V_Cost (Remaining_Delta_V left);
    arrival: Time_of_Arrival" ];
```

Algorithm 11: Graphviz syntax.

Graphviz identifies nodes by the text string within them, so it was straightforward to map asteroids to nodes and transfers to edges within the program. If we have multiple edges going into the same node, we know the asteroid was visited multiple times.

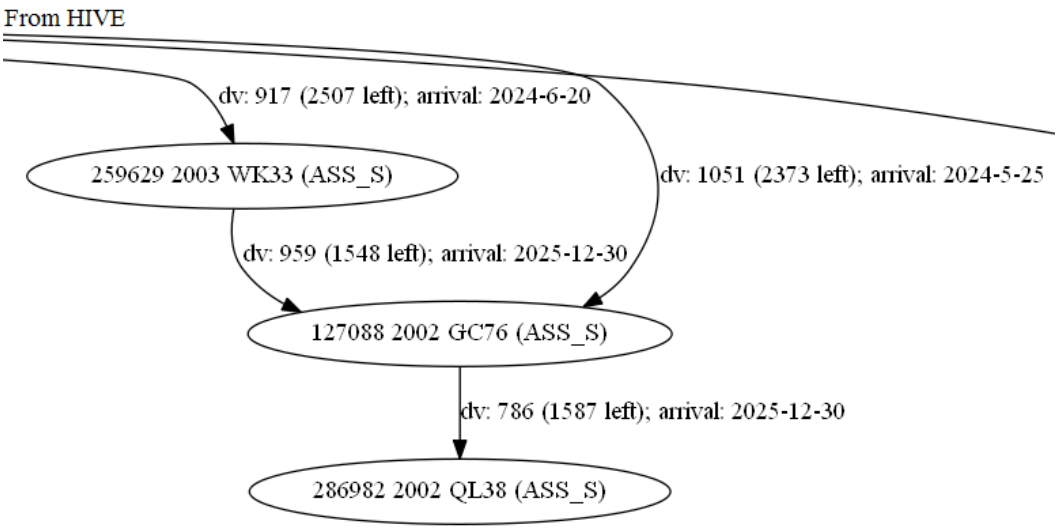

Figure 41: Cutout from a Graphviz representation of showing two BEEs exploring the same asteroid.

Graphviz, with the default configuration, produced a tree graph. In our case, the top node was the HIVE, which had 12 transfers leaving it. Each of those transfers arrived at an asteroid. Ideally, there should be 12 separate asteroids on the second level; though multiple BEEs occasionally visited the same asteroid. The third level consisted of the second asteroid in each BEE's route. Should multiple BEEs visit the same asteroid, its location was set by the last route position which visited it. In Figure 41, we see that one of the BEEs travelled directly to 2002 GC76, while another went there via 2003 WK33, so GC76 ended up in the third row, pushing 2002 QL38 into the fourth.

Appendix H Selected bugs

Software never works perfectly on the first try. There are always some small errors or components which do not work as expected. Most are relatively minor and easy to track down, but occasionally we come across a bug that proves challenging to resolve. Instead of documenting these events as they occur, which would have cluttered the text with occasional digressions, we have chosen to gather them here in this appendix.

H.1 Lambert's Problem Returns NaN When the PyKEP::Planets Align

We spent most of our first day of coding trying to get Lambert's Problem to return an actual orbit, rather than a pair of vectors filled with “not a number”. What we eventually discovered was that the test transfer we were using, a simplified Hohmann Transfer[40], was an edge case that common solutions to Lambert's problem were poorly equipped to handle.

In an idealised Hohmann Transfer, the departure position and arrival position are exactly half an orbit apart. This means that the Sun and these two positions form a line in space. Lambert's Problem, meanwhile, relies on working within the plane defined by the triangle formed by those three points. Naturally, when all three points are aligned, that plane is undefined and the whole method fails.

Our solution was to wrap the Lambert's Problem solver in a method which checked for such alignments, and to calculate the delta v of such situations through other means. Specifically, we could calculate the Hohmann transfer cost, since the reason we were testing with Hohmann in the first place is that it is straightforward to calculate the delta V cost of a Hohmann transfer.

H.2 Auto Versus Auto&

During testing of the resume functionality, we were surprised by the slow performance. As the overhead of the search was close to 0% due to the cost of solving Lambert's problem, we expected the program to resume execution in virtually no time at all. Some time would be required to read and process the file, but the rest should appear instantaneous. What we experienced was nowhere near instantaneous.

To determine whether a pair of asteroids had already been computed, we had to first lookup a map of all the destination asteroids for the given origin asteroid, and then find the specific destination asteroid in that map. It turned out we had misunderstood a new feature of *C*++11. It includes the keyword auto as a shorthand for manually specifying the type of a variable.

During this process, we temporarily stored a reference to the second map, with a type of auto. What we failed to understand is that auto always denotes a value type. Although the find function of the map returned a reference to another map, it was stored by value. Because of the size of the map, likely best measured in kilobytes, and the high frequency of lookups, this slowed down execution tremendously.

The performance problem was solved by adding an ampersand after the auto keyword for the affected lines of code, making them auto& rather than auto. The performance increased by roughly three orders of magnitude, to what we had expected.

H.3 Eclipse Crashes When Printing a Certain Line

We were using the IDE Eclipse during development of the AI Simulator. For some reason, one line of Java code ended up causing Eclipse to crash:

```
System.out.println("\t" + name + "\t" +id);
```

Removing the line allowed both Eclipse and the Simulator to run successfully (and with flawed results). Debugging the problem was out of the question, as the debugger would not start when the line was included. Some careful and slow experimentation showed that it was the name string that could not be printed. We never discovered the reason, but suspected some memory corruption. The strange part is, it should be difficult to accomplish this in Java and, even if the code was flawed, Eclipse should not have crashed when running it.

H.4 HIVE is not Hive

At one point, the file format of an old Transfer Map file was converted to the new one. The process itself finished seemingly without any errors, but the Transfer Map Generator was unable to resume from the new file. It immediately quit during resume because it could not find an asteroid with the name *"HIVE"* in the transfer index. This seemed strange, because if the *HIVE* was not included in the index, how could any other asteroid be included. It was, after all, the origin of all origins in the search.

Debugging revealed that there was indeed an asteroid named *"Hive"* in the Transfer Map file, and that it was structured correctly. If so, why could the search not find it? Careful inspection revealed that the asteroid called *"Hive"* had changed name to *"HIVE"* during development, causing the search to look for *"HIVE"* while all it could find was *"Hive"*. The conversion process was amended by including a duplicate entry in the transfer index of files from the initial file format. With the change, both *"HIVE"* and *"Hive"* pointed to the same data, fixing the problem.

H.5 Sorting in the Right Order

The first few results from our genetic algorithm were surprising to say the least. Not only were the solutions poor, they seemed to get worse the more information they had available. This suggested that there was something terribly wrong with our evolution. While it is obvious, in hindsight, that what we were seeing was an attempt to minimise fitness, we initially assumed the issue was in the assignment of fitness scores to asteroids.

There was, indeed, an issue there. As we had not yet set the scores for the individual asteroids, every asteroid had the default -1 score. Meanwhile, however, the asteroid class scores had been set and were in the 20-30 range. Which meant that until about 20 asteroids of a given class had been visited, the total value of the asteroid remained positive. So while we thought our issue was due to the AI trying to maximise a negative fitness, it was in fact minimising a positive one.

We eventually traced the problem to the selection code. All our selection methods were sub-classes of an abstract selector class, which among other things held a custom comparator for use with ArrayList's built in sort() method. The comparator was written perfectly to specification, but assumed that the sort() method sorted data with the largest value first.

Interestingly enough, which order the sort method actually sorts things in is not particularly well documented. We have not found any definitive statement from the javadoc which specifies which order it uses. But by experimentation we found that if we reversed the comparator, the code once again tried to maximise fitness. The evolvers still produced lower utilities with more information but, as discussed in chapter 8, there were other explanations for that.

H.6 Fitness Evaluation and Class Value

In order to reuse as much code as possible, both the No Communications and the Stigmergy used simulation code to evaluate the fitness of a genome. This included a method intended to keep track of the current point value of each asteroid class. This method made persistent changes to the

simulator, decreasing the value of the class each time it was called (based on the function described in section 4.3).

This meant that after the first few generations, the class value of any given class had been reduced to the point where it was effectively zero. As a result, the evolutionary algorithm had a fitness function where visiting one or two asteroids with moderate to high asteroid specific values provided more fitness than exploring three low priority asteroids. The solution was to reset the simulation more thoroughly between each genome evaluation, which led us straight to the second problem.

The Stigmergy fitness class served double duty, functioning both as a fitness evaluator for genomes and a repository of the information known to the evolver at the time. This meant that we could not simply reset the simulation, since it contained vital persistent information about the environment. After some consideration, we ended up rewriting most of the genome evaluation code so that it would pass along its own persistent information as method variables rather than storing it in the object itself.

H.7 Math.random

When Java programmers need (pseudo)random numbers, they usually turn to Java's “Math.random”. This is normally not a problem, not even for multiple threads working concurrently. However, a problem arises when multiple threads have to create large amounts of random numbers. If the previously mentioned strategy is used, all threads end up using the same object, which causes a bottleneck due to locking. To overcome this, we initialised multiple random number generators, one for each thread. These were then passed as method arguments wherever random numbers were needed. This improved concurrent performance significantly on our dual core development computers.

11 Bibliography

1. Verhoeven, C. J. M., Bentum, M. J., Monna, G. L. E., Rotteveel, J. & Guo, J. (2011) 'On the origin of satellite swarms', Acta Astronaut, 68, pp. 1392-1395.

2. Rouff, C. A. (2002) 'Autonomy in future space missions', Proc. IEEE Aerospace 788 Conference

3. Curtis, S. A., Truszkowski, W., Rilee, M. L. & Clark, P. E. (2003) 'ANTS for human exploration and development of space', Aerospace Conference, 2003. Proceedings. 2003 IEEE, 1, pp. 1-261.

4. D'Arrigo, P. & Santandrea, S. (2006) 'APIES: A mission for the exploration of the main asteroid belt using a swarm of microsatellites', Acta Astronaut, 59, pp. 689-699.

5. Stuart, J. R., Howell, K. C. & Wilson, R. S. (2014) 'Design of End-To-End Trojan Asteroid Rendezvous Tours Incorporating Potential Scientific Value', AAS/AIAA 24th Space Flight Mechanics Meeting, Santa Fe, New Mexico.

6. Elvis, M. & Esty, T. (2014) 'How many assay probes to find one ore-bearing asteroid?', Acta Astronaut, 96, pp. 227-231.

7. Tripp, H. & Palmer, P. (2010) 'Stigmergy based behavioural coordination for satellite clusters', Acta Astronaut, 66, 1052-1071.

8. Carr, J. J. (1993) 'Directional or omnidirectional antenna', Joe Carrs Receiver Antenna Handbook. Available at http://146970.com/PDFs/Antenna - Directional or Omnidirectional Antenna.pdf (Accessed 2015-03-25).

9. Birlan, M. (2000) 'Dynamic and physical considerations on the asteroids density', Earth Moon Planets, 88, pp. 1-10.

10. Carry, B. (2012) 'Density of asteroids', Planet. Space Sci, 73, pp. 98-118.

11. Britt, D. T., Yeomans, D., Housen, K. & Consolmagno, G. (1987) 'Asteroid density, porosity, and structure', Asteroids III

12. Ahrens, T. & Harris, A. (1992) 'Deflection and fragmentation of near-Earth asteroids'.

13. Ross, S. D. (2001) 'Near-earth asteroid mining', Space Industry Report.

14. Yano, H. et al. (2006) 'Touchdown of the Hayabusa spacecraft at the Muses Sea on Itokawa', Science, 312, pp. 1350-1353.

15. Tsuda, Y., Yoshikawa, M., Abe, M., Minamino, H. & Nakazawa, S. (2013) 'System design of the Hayabusa 2—Asteroid sample return mission to 1999 JU3', Acta Astronaut, 91, pp. 356-362.

16. Gal-Edd, J. (2014) 'The OSIRIS-REX Asteroid Sample Return – Mission Operations Design', American Institute of Aeronautics and Astronautics, [Online] DOI:10.2514/6.2014-1721

17. Rayman, M. D., Fraschetti, T. C., Raymond, C. A. & Russell, C. T. (2006) 'Dawn: A mission in development for exploration of main belt asteroids Vesta and Ceres', Acta Astronaut, 58, pp. 605-616.

18. Kepler, J. (1609) Astronomia nova.

19. Moore, W. (1813) A Treatise on the Motion of Rockets. To which is added, An Essay on Naval Gunnery.

20. Avendaño, M. & Mortari, D. (2010) 'A closed-form solution to the minimum Delta V^2 tot Lambert's problem' Celestial Mechanics and Dynamical Astronomy, 106, pp. 25-37.

21. Lucas, G. (1980) Star Wars - Episode V: The Empire Strikes Back.

22. Bowell, E. (1994) 'astorb.dat', The Asteroid Orbital Elements Database. Available at ftp://ftp.lowell.edu/pub/elgb/astorb.html (Accessed 2015-02-26).

23. Badescu, V. (2013) Asteroids.

24. Pyke, G. H. (1984) 'Optimal Foraging Theory: A Critical Review'.

25. Pierce, G. & Ollason, J. (1987) 'Eight reasons why optimal foraging theory is a complete waste of time'.

26. Stuart, J., Howell, K. & Wilson, R. (2014) 'Application of Multi-Agent Coordination Methods to the Design of Space Debris Mitigation Tours'.

27. Laporte, G. (1991) 'The Vehicle Routing Problem: An overview of exact and approximate algorithms'.

28. Eksioglu, B., Vural, A. V. & Reisman, A. (2009) 'The vehicle routing problem: A taxonomic review', Computers & Industrial Engineering, 57, pp. 1472-1483.

29. Back, T. & Schwefel, H.-P. (1993) 'An overview of evolutionary algorithms for parameter optimization'. Evolutionary Computation, 1, pp. 1-23.

30. Floreano, D. and Mattiussi, C. (2008) Bio-inspired artificial intelligence: theories, methods, and technologies.

31. Back, T. (1994) 'Selective pressure in evolutionary algorithms: A characterization of selection mechanisms', Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference, pp. 57-62.

32. Tripp, H. & Palmer, P. (2010) 'Distribution replacement for improved genetic algorithm performance on a dynamic spacecraft autonomy problem', Optimization and Engineering, 42, pp. 403-430.

33. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. (1953) 'Equation of State Calculations by Fast Computing Machines', The Journal of Chemical Physics, 21, pp. 1087-1096.

34. Nelder, J. A., & Mead, R. (1965) 'A simplex method for function minimization', The Computer Journal, 7, pp. 308-313.

35. (Anonymously) “lflem...@gmail.com” (2007) 'nelder-mead-simplex'. Available at: https://code.google.com/p/nelder-mead-simplex/ (Accessed 2015-03-10).

36. Bentea, L. (2005) 'CodeCogs: Nelder'. Available at: http://www.codecogs.com/library/maths/optimization/nelder.php (Accessed 2015-03-10).

37. Hutt, M. (2015) 'Nelder-Mead Simplex Method'. Available at: http://www.mikehutt.com/neldermead.html (Accessed 2015-03-10).

38. Morr, S. (2012) 'Nelder-Mead optimizer'. Available at: https://github.com/blinry/nelder-mead-optimizer (Accessed 2015-03-10).

39. Yang, G. and Vanroose, P. (2012) 'Vision Numerics Library'. Available at: http://vxl.sourceforge.net/ (Accessed 2015-03-10).

40. Hohmann, W. (1925) The Attainability of Heavenly Bodies.

41. 'OpenMP'. Available at: http://openmp.org/wp/ (Accessed 2015-03-15).

42. Izzo, D. and Biscani, F. (2015) 'PyKEP Library Website'. Available at: http://esa.github.io/pykep/index.html (Accessed 2015-03-09).

43. Haas, J. (2015) 'hexdump', Available at: http://linux.about.com/library/cmd/blcmdl1_hexdump.htm (Accessed 2015-03-25).

44. 'Graphviz'. Available at: http://www.graphviz.org/ (Accessed 2015-03-25).

45. NASA (2005) 'The Revolving Door to Mars'. Available at: http://mars.jpl.nasa.gov/spotlight/porkchopAll.html (Accessed 2015-03-25).

46. (Anonymously) “Flappiefh”, “Autiwa”, & “Mizusumashi” (2008) 'Asteroid Belt.svg'. Available at: http://commons.wikimedia.org/wiki/File:Asteroid_Belt.svg (Accessed 2015-03-25).

47. Mothediniz, T. (2003) 'Distribution of taxonomic classes in the main belt of asteroids', Icarus, 162, pp. 10–21.

48. Minor Planet Center 'New- And Old-Style Minor Planet Designations'. Available at: http://www.minorplanetcenter.net/iau/info/OldDesDoc.html (Accessed 2015-07-03).