Aerospace Safety Requirements Allocation

Final Report

Submitted for the BSc in
Computer Science with Industrial Experience

May 2016

by

Radu Diaconeasa

Word Count: 8908

Acknowledgements

I would like offer my special thanks to Dr Yiannis Papadopoulos for his supervision throughout the Final Year Project and for the constructive feedback provided in the Initial Report. I would also like to thank Dr David Parker for his valuable suggestions during the Interim Demonstration session which have been highly appreciated.

Finally, I am particularly grateful for the assistance given by Ioannis Sorokos in implementing the important parts of the project.

Table of Contents

1 Introduction

The domain of this project is safety critical systems, in particular, Aerospace Systems. These systems have high safety requirements and they are governed by complex lifecycles which are in many cases ruled by standards.

One of the recent standards is the ARP (Aerospace Recommended Practice) standards which define how the development of these systems should be done in a way that safety requirements can be met by the design of the systems. As part of this development, one of the problems that the standard defines and tries to address is the progressive allocation of requirements (S-18, SAE, 2010).

At the early stages the standard assumes that the requirements are captured through the process of risk analysis at the level of the system of the aircraft. As architectures for the aircraft are being developed safety requirements are allocated to elements of the architecture – subsystems and then components.

However, this process is manual in the standard and recently there has been a lot of research to automate it. A similar situation also exists in the automotive industry. International Organisation for Standardization (ISO) 26262 (TC 22/SC3, 2011) standard describes a not very dissimilar process for allocation of requirements in the context of the development and the composition of the system. There is a lot of research in the general topic of the allocation of requirements and some of the research has been focused on DALs (Development Assurance Level) which are applicable in the Aerospace Industry.

In order to capture safety requirements, in this industry they use the concept of Development Assurance Level. These are qualitative levels which saw a different level of assurance achieved. Usually the standard defines five levels from either 0 to 4 or from A to E in the Aerospace Industry and each level is associated with a quantitative target to be achieved in terms of the probability of failure on demand. Because systems can contain software, the integrity levels are associated with development processes. Therefore, for high integrity software (i.e. level A) the standard prescribes formal methods while for low integrity software (i.e. level E) there is much more flexibility involved.

This project looks to extend state-of-the-art research that aims to optimize the process of assigning safety requirements in complex systems within the aerospace industry.

In the next sections, this research based report presents the analysis and compares the efficiency of three solutions to this problem – Genetic Algorithm, Tabu Search and Ant Colony Algorithm (with different approaches in the pheromone distribution).

2 Aim and Objectives

To develop a solution for the automated allocation of requirements in the Aerospace Industry through the exploration of possible improvements via application of metaheuristics and parallel exploration of the search space in many places

The above aim of the project will be met by the following numbered objectives:

Objective 1 – Analyse existing methods of DAL allocation applied in other scenarios

Evaluate past research on other similar safety requirements allocation problems and understand the effectiveness of various relevant metaheuristics.

Objective 2 – Develop Ant Colony Algorithm

To develop a solution based on the effective permutations of the basic Ant Colony algorithm that will solve the problem.

Objective 3 – Apply to a case study

Implement and adapt the developed algorithm to fit into a case study, in particular the DAL allocation in the Aerospace Industry, taking into account the principles and constraints of this scenario.

Objective 4 – Compare performance of Ant Colony to other solutions reported in the literature

To analyse the efficacy of the developed algorithm by comparing various permutations of it and in relation with the previously implemented Brute Force and Tabu Search Algorithm.

Objective 5 – Connection of the developed software to the HiP-HOPS tool

To incorporate the solution into the HiP-HOPS model based safety analysis optimization framework thus allowing to start from the model, run the optimization, get the results back and, if they are not satisfactory, redo the model, change the analysis and eventually get different results.

3 Background

3.1. Problem Context

In the Aerospace Industry, safety and reliability are critical. Safety is used to refer to the capability of the system to prevent any hazards happening where people and environment can be affected negatively. Reliability represents the capability of the system to correctly perform its intended functions when requested. There are high concerns that various components of the aircraft can fail thus producing a disaster. However, Figure 1 below shows that the continuous efforts focused on the above concepts have led to a significant decrease in aircraft accidents in the last couple of decades:

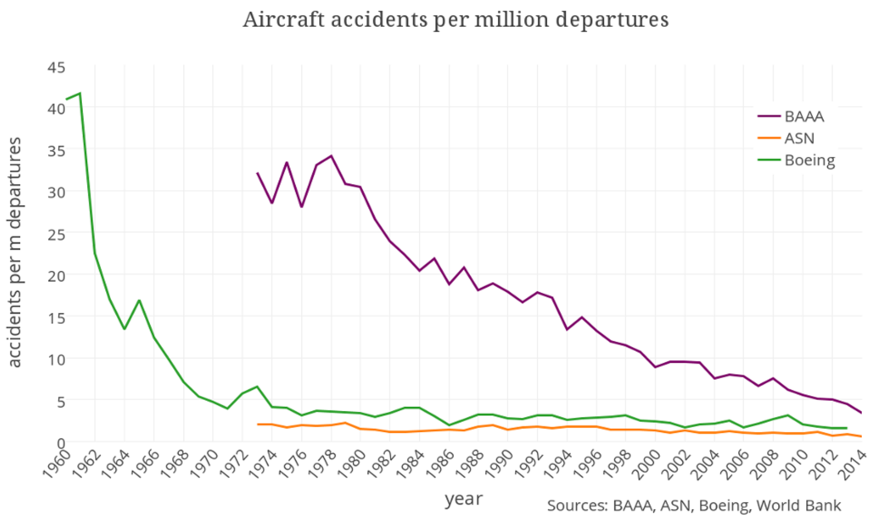


Figure 1 – Aircraft accidents per million departures (Statistical Summary of Commercial Jet Airplane Accidents, 2015)

Although the aircraft accidents statistics show a major improvement in the safety and reliability of the aircraft systems, the software used to support various functions of the modern aircrafts has grown significantly in complexity in the recent years as suggested in the Figure 2 below:

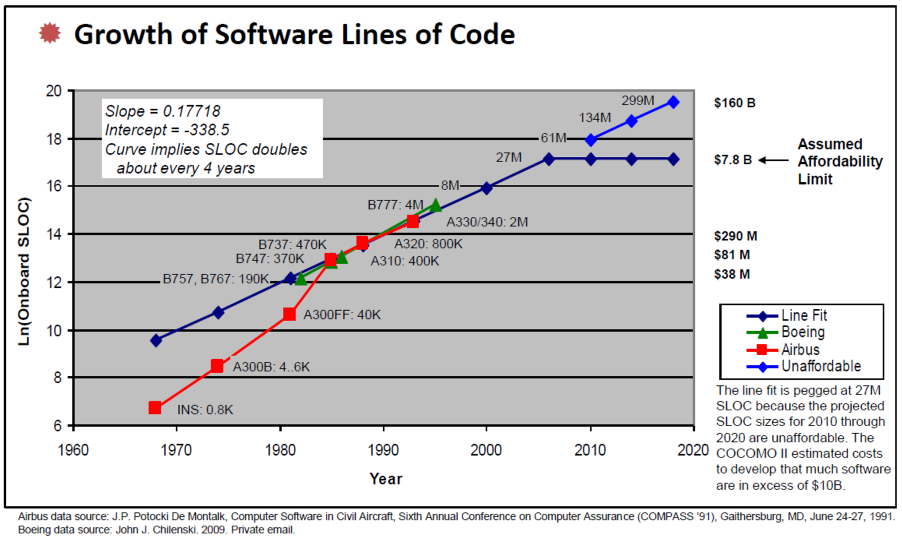


Figure 2 – Growth of Software Lines of Code (EngineeringNewWorld.com, 2014)

As systems become more complex, software cannot be rigorously tested any longer and this fuels the growing concerns that, in the future, software might not be reliable enough to perform the various functions of the aircraft. System requirements identification and allocation are normally gathered in the early stages of systems engineering and are therefore crucial in assuring that the system architecture and design are built in a way that these requirements can be met.

With the modern aircrafts nowadays it is much harder to understand the dependability of the components and the manual analysis has become infeasible. Figure 3 below presents a rather complicated diagram of an aircraft Hydraulic System Power Distribution with many interdependencies between components. If, for instance, System A Reservoir fails as illustrated in the below figure it would be significantly harder to predict, upon a manual exploration of the system, what implications this would have on other components (i.e. Trailing Edge Flaps):

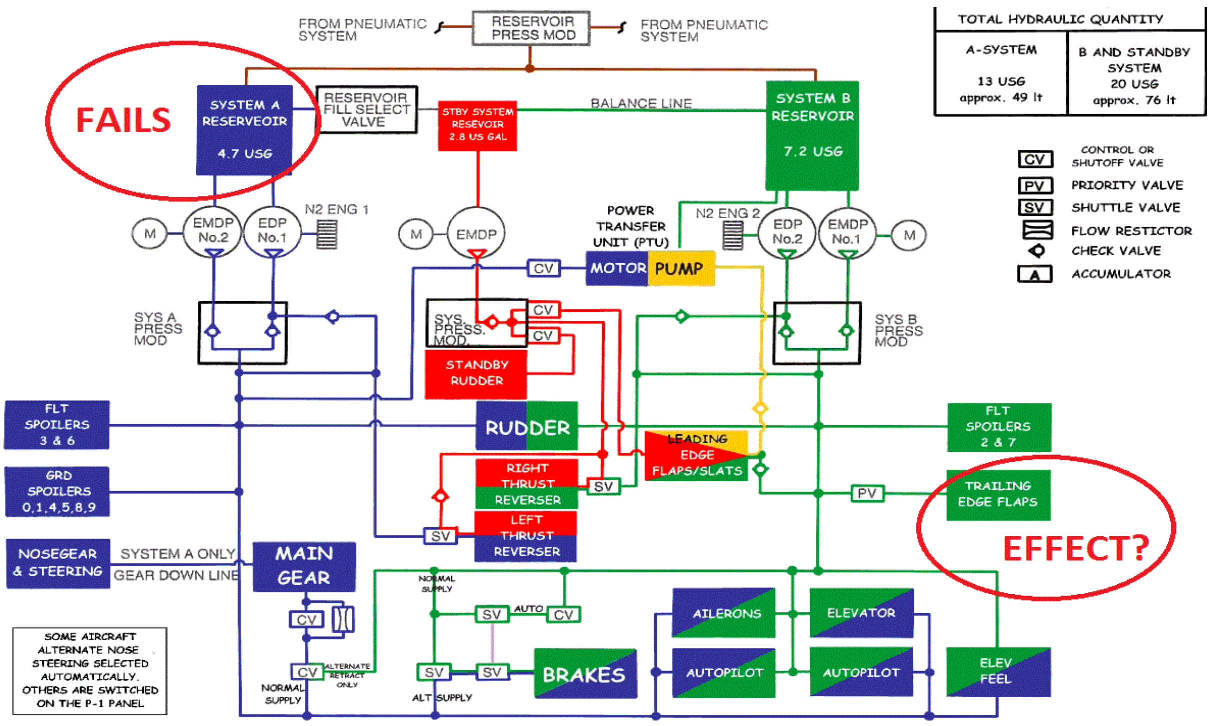


Figure 3 - Aircraft Hydraulic System Power Distribution (AircraftEngineering.wordpress.com)

Considering the system requirements, hazards of the systems are assigned certain integrity levels which must be fulfilled in order to prevent these malfunctions happening with a certain level of integrity. In the aviation industry, these hazards refer to the potential for harm caused by various technical aspects within the aircraft system which could lead to human injuries, damage or inability to perform a predefined function (Maragakis, et al., 2009). Once the architecture is designed, safety requirements that satisfy the integrity levels are assigned.

This is a rational way of thinking about developing a system as we need to clearly identify which components contribute to which failures in the system and on the base of this analysis we can define the criticality of the components. For instance, if a component contributes to a failure which is very critical, such as failure to open the landing gear or even engine failure, then it has to inherit a high integrity requirement. Similarly, if it does not contribute to any failure it will be assigned a lower integrity requirement.

In the allocation of safety requirements, various standards provide recommendations on how these requirements should be allocated to subsystems and components. For instance, for the automotive industry, ISO26262 (TC 22/SC3, 2011) standard comes with the concept of ASIL (Automotive Safety Integrity Levels) while in the aerospace industry the ARP4754-A (S-18, SAE, 2010) standard defines DALs (Development Assurance Level).

DALs are determined early and assigned to system-level safety functions via risk analysis (Papadopoulos, 2015). These are allocated onto the system's components using a process called DAL decomposition in which components in a subsystem can be assigned lower DALs and, in combination with other components or elements, can still satisfy the DAL of its parent function. DALs are defined through five levels (ARP4761, 1996) depending on the severity of the components as seen in Table 1 below:

Severity	Catastrophic	Hazardous / Severe-Major	Major	Minor	No Safety Effect
DAL	A=4	B=3	C=2	D=1	E=0

Table 1 – DAL to Severity equivalence

The decomposition rules are more restrictive in the aerospace compared to the automotive industry due to higher safety constraints in the first one. They use the concept of Functional Failure Sets (FFS) which contains the components or minimal combinations of components, also known as Functional Failures (FF), that need to fail in conjunction in order for the whole system/function to fail (ARP4754-A, 2010). Normally, these elements of the architecture inherit the DAL from their parent system but the standard allows for some flexibility in the allocation of the DALs, thus allowing for some components to have a lower integrity and therefore a lower cost. Given a FFS with a DAL of k we can either:

Option 1: have only one component assigned a DAL of at least **k** and the rest of the components of at least **k-2**

Option 2: two components assigned a DAL of at least **k-1** and the other components of at least **k-2** (Sorokos, et al., 2015).

Overall, in the aerospace industry there are fewer options available with regards to the allocation of safety requirements. Although this should simplify the allocation process, in a complex system such as the Hydraulic System Power Distribution illustrated in Figure 2 earlier it would still require the evaluation of hundreds of thousands of possible combinations in order to find the optimal, most cost-effective solution. A cost-effective solution both satisfies the allocation rules above and it lowers the costs compared to having DAL A assigned to all components. However, such a process, executed manually, would not only require high amounts of time but would also return ineffective results with many components being allocated a higher DAL than required and therefore incurring unnecessary costs (Sorokos, et al., 2015).

Through the automation of this process it is hoped that the dependability analysis is simplified and optimal DAL allocations are effectively achieved in complex systems thus minimizing the total development costs. Whichever component or function has, for instance, DAL A, it will require higher development costs due to its higher integrity level. However, if a subsystem is assigned DAL A, not all components in that subsystem need to inherit its DAL (a more detailed example of how lower DALs can be achieved applying the decomposition rules is provided in the next sections). Significant development time and effort are also saved for more important issues while the process can be easily repeated with every change in the subsystem as the system design evolves.

3.2 Previous work

There have been some efforts recently to automate this allocation of requirements as presented below:

3.2.1 HiP-HOPS

While similar research has been done to enable the DAL allocation starting from FFS (or Minimum Cut Sets) in (Bieber, et al., 2011), HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) (Papadopoulos, et al., 2011) is a tool which addresses the above issue by automatically building a fault tree starting from the system model annotated with failure behaviour information and by analysing the dependency of the faults through failure logic propagation (Sharvia & Papadopoulos, 2011). A fault tree illustrates the failures in the form of a tree where the main failure event sits at the root while the leaves represent the base component failures (Walker, 2009). The elements in the fault tree are linked through logical connectors (i.e. AND and OR). Figure 4 below shows an example of how failures are distributed and connected in a fault tree:

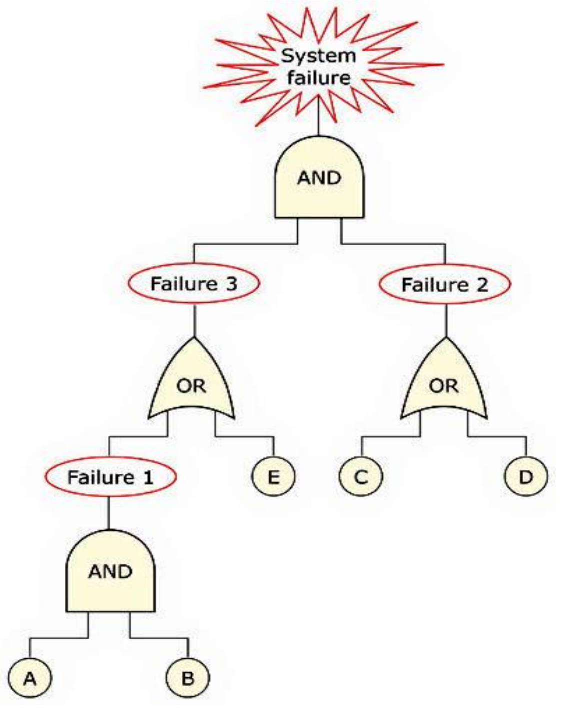


Figure 4 – Fault tree example (Hobbs, 2016)

Once the synthesis of the fault trees is complete, the method also produces the FFS (also known as Minimum Cut Sets) which illustrate the components or minimal combinations of components that need to fail in conjunction in order for the whole system/function to fail. These can be used in the allocation of DALs onto the system's components using DAL decomposition rules. Figure 4 above can be interpreted as follows:

- In order for the whole system to fail, both Failure 2 and Failure 3 are needed.
- Failure 2 takes place if either component C or D fails.
- Failure 3 takes place if either component E fails or Failure 1 takes place.
- In order for Failure 1 to occur, both component A and B need to fail.

The following example is based on Yiannis Papadopoulos' Honeywell presentation (Papadopoulos, 2015):
Let us consider the following scenario where a system S is designed to have 2 functions X, Y and each function has two malfunctions – O (Omission or Loss) and C (Commission or Unintended delivery). For the purpose of this example we will also assume that the criticality analysis has shown that DAL of X(O) = **DAL B**, X(C) = **DAL D**, Y(O) = **DAL A** and Y(C) = **DAL B**. Figure 5 below is a representation of the system S architecture and Failure Modelling used by HiP-HOPS in this example with six components from K1 to K6 with AND and OR operators being replaced with **.**(dot) and **+** (plus):

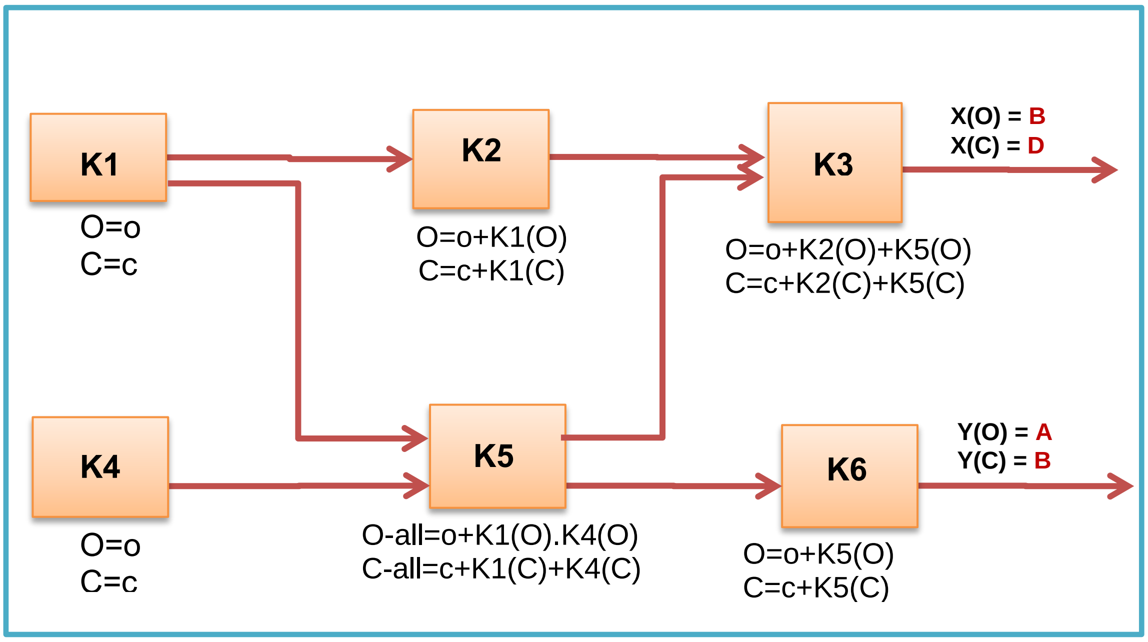


Figure 5 – System Architecture and Failure Modelling

By the failure modelling design in the above figure, K3 is going to fail if either K2 or K5 fails. Similarly, all the other components are designed this way. The fault trees for the four functional failures and their cut sets are as follows:

- **[1]** X(O) = K3o + K5o + K2o + K1o; DAL = **B**
- **[2]** X(C) = K3c + K5c + K4c + K2c + K1c; DAL = **D**
- **[3]** Y(O) = K6o + K5o + K4o . K1o; DAL = **A**
- **[4]** Y(C) = K6c + K5c + K4c + K1c; DAL = **C**

From equation [3] it results that K6 and K5 will inherit the parent DAL because these FFS have only one element and therefore they will be assigned at least A. Between K4 and K1 there is an AND operator which means that, after applying the 2 options recommended by the standard, [K4,K1] can either have DAL [A,C] or [C,A] or [B,B].

Equation [4] suggests that K6, K5, K4 and K1 need to have at least DAL C (inherited from parent). These results can be represented as:

[3] = > K6=**A**, K5=**A**, [K4,K1]=[**A,C**] or [**C,A**] or [**B,B**] **[5]**
[4] = > K6=**C**, K5=**C**, K4=**C**, K1=**C** **[6]**

From [5],[6] it results that all of the components still have the same DALs while [1] suggests that K3, K5, K2 and K1 have at least DAL B (inherited).

[5],[6] = > K6 =**A**, K5=**A**, [K4,K1]=[**A,C**] or [**C,A**] or [**B,B**] **[7]**
[1] = > K5=**B**, K1=**B**, K3=**B**, K2=**B** **[8]**

From [7] and [8], [K4,K1] can only have [B,B] due to K1 who has DAL B in equation 1. Finally, equation [2] shows that K1 to K5 have inherited DAL D from their parent function. Therefore:

[7],[8] = > K6=**A**, K5=**A**, K4=**B**, K1=**B**, K3=**B**, K2=**B** **[9]**
[2] = > K5=**D**, K4=**D**, K1=**D**, K3=**D**, K2=**D** **[10]**

Equations [9] and [10] give the final allocations of DALs as follows:

[9],[10] = > K6=**A**, K5=**A**, K4=**B**, K1=**B**, K3=**B**, K2=**B**

The above example demonstrates that more economical solutions than "DAL A to ALL" approach can be generated. However, this analysis is manual which makes it difficult to return effective results in more complex systems but various metaheuristics can be implemented into HiP-HOPS and automate this process.

3.2.2 Tabu Search Algorithm

Tabu Search (Glover, 1986) is a metaheuristic optimization technique and its principles refer to the storage of recently evaluated candidate solutions which are not eligible for generation of further candidates. These candidates are stored in a short-term memory structure which is usually called the "Tenure".

As many other metaheuristics, Tabu Search is an exploratory algorithm which means that the globally optimal solution is not guaranteed to be found. The algorithm drives the search towards both nearby optimal solutions ("Intensifying") and new regions by hiding the potential globally optimal solutions ("Diversifying"). Also, it brings the concept of "Aspiring" candidates for those which beat the current best solution. Tabu Search initially generates a random solution out of the potential ones then, through an iterative process, it generates a random number of candidates from the current one, it selects the first one in the order of cost, provided that they are not "Aspiring" or found in the "Tenure", it replaces the current best solution if the candidate beats it and finally the Tabu Tenure is updated with the new candidate while the oldest one is removed.

3.2.3 An extension to HiP-HOPS using Tabu Search Algorithm

In his work, Ioannis Sorokos (Sorokos, et al., 2015) is considering an extension of the dependability analysis and optimisation tool, HiP-HOPS, and the case study chosen is based on the model of an aircraft wheel-braking system. The aim of the research is to extend previous work in order to enable a more efficient, model-based automation of safety requirements allocation. The initial paper his research is based on is the automotive SIL allocation (described below) and the DAL allocation from minimum cut sets in (Bieber, et al., 2011). However, it is stated that the new approach here is the way the allocation process starts from the system model rather than the minimum cut sets found in (Bieber, et al., 2011). Therefore, this work, through the implementation into HiP-HOPS, allows the synthesis of the fault tree generated from the system model which is then analysed to produce the Minimum Cut Sets.

Although several optimization algorithms were available, Tabu Search was considered a good candidate on the basis of its proven good performance in previous work such as the allocation of automotive safety requirements (Azevedo, et al., 2013). The research suggests that Tabu Search Algorithm was efficient in solving the DAL allocation problem. The algorithm was applied on a relatively small scale of the search space with 531,441 possible allocations but the results showed that 1000 iterations of Tabu Search were executed in less than a second while the optimal solution was found in 5.6 milliseconds on average.

3.2.4 Genetic Algorithm

Similar efforts were put to automate the allocation of safety integrity levels in the automotive industry. Luis Azevedo (Azevedo, et al., 2013) has published a paper in this regard where a penalty-based genetic algorithm was adopted and applied to a hybrid braking system to analyse its efficacy.

A Genetic Algorithm (GA) (Holland, 1975) is a search heuristic which tries to simulate the natural selection process. This involves finding the fittest candidate that survives over several generations for an individual problem. In nature, the fight for resources will always determine the strongest candidate which dominates over all the others. GA is concerned with making sure that lower quality candidates are not considered and stronger candidates are evaluated further based on probabilities.

With every generation, better candidates are hoped to be produced while the ones which are less fit die out. At some point, stagnation (also known as convergence) can take place when no different candidates are produced any longer. This should not necessarily be regarded negatively as normally candidates become very similar and reach the fitness level required after some time. However, early convergence is not desired as the solution found might not be up to the expected standard. This phenomenon can be avoided through various techniques for generating diversity.

3.2.5 ISO 26262 - Automotive Safety Integrity Levels (ASIL)

ISO 26262 is a standard which defines safety requirements for the automotive industry in a similar way as ARP4754a standard does for the aerospace industry with the caveat that DALs are replaced with ASILs and the decomposition rules are less stringent.

In his paper, Luis Azevedo uses the penalty-based genetic algorithm (Coit & Smith, 1996). The concept of “penalty” is used in differentiating two components which have the same ASIL cost by comparing the number of safety requirement violations for each of them. Such violations occur when the sum of ASILs assigned to the failure modes in the cut set does not meet the total required ASIL. As violations will make a component appear more expensive, the counter called “invalid_ASILs” will therefore determine which of the components is less fit.

New candidates are generated from the old ones using two genetic operators – mutation and recombination (also known as crossover). While the mutation operator tries to avoid convergence by encouraging the exploration of further solutions, the recombination operator will focus on generating candidates from the fittest “parents”. However, an effective usage of these two techniques can avoid the early convergence issue mentioned above.

Results show that although the genetic algorithm used in this work did not always retrieve the global optimal solution, low cost solutions were still returned in those cases. Furthermore, the performance of the algorithm seems to be high and it is claimed that the analysis is normally performed in less than a second.

3.2.6 Linear Programming - DALculus

DALculus (Bieber, et al., 2011) comes with a different approach and a different way of looking for solutions. They use linear programming to formalize the DAL allocation rules as constraints. Their constraint solver uses both the allocation rules recommended by ARP4754a and the user defined constraints in order to find the most relevant and efficient solutions.

It is also noted that in order to find independence relations between components so that the qualitative requirements can be allocated, the Constraint Satisfaction Problem (CSP) needs to be solved. They define CSP as a set of constants and variables representing either user inputs, constraints or Minimum Cut Sets. The CSP is then used in the DAL allocation problem with the DAL allocation rules being translated into various mathematical functions and equations to be solved.

DALculator, which is the tool used to identify independencies and allocate DALs. Although the solutions produced by DALculator are not always optimal, its particular advantage is that it can receive instructions or directives from the user such as specifying whether 2 components should be independent or not. Therefore, it not only supports the user in finding different independence relations and DAL allocations but also enables the tool to verify whether an existing allocation is fit for the requirements or not.

However, while this approach works for relatively simple systems, because it is based on and starts from the Minimum Cut Sets, MCS can become too complicated for the constraint solvers as the system grows in complexity. This also represents one of the reasons why, in this report, metaheuristics have been considered in the DAL allocation instead of linear programming.

3.3 Ant Colony Optimization (ACO)

Although research has been done and various metaheuristics have been used, not all of them have been considered. In the Aerospace Industry, the systems can go to a very high scale and complexity which means that there is always space to improve and to explore the performance of other metaheuristics which is what this project is trying to achieve.

As described in the previous section, in the ASIL allocation problem used in the Automotive Industry various metaheuristics have been used while in the DAL allocation problem brute force and the Tabu Search approach have been implemented. However, the Ant Colony Algorithm is a relatively new alternative and has not been used in this particular problem.

As many other algorithms, Ant Colony is a metaheuristic which is based on the exploration of a space of solutions. Because metaheuristics are not required to search the entire space for a solution means that the global optimal solution is not always guaranteed to be found which one of the drawbacks is. However, the automated nature of this process still offers an advantage over the manual search for solutions especially in large scale problems. The drawback can be avoided through evaluations which provide guidance in searching the space effectively.

Compared other metaheuristics (i.e. tabu search, simulated annealing, evolutionary computation), Ant Colony Optimization is unique through its characteristics - *a constructive, population-based metaheuristic which exploits an indirect form of memory of previous performance* (Dorigo, M., 2004).

3.3.1 Real and artificial ants

Ant Colony Algorithm simulates the ant colonies behaviour in real life (Dorigo, M., 2004). Researchers have been inspired by the highly structured social organization of the ants and they have tried to replicate their behaviour in various optimization problems.

Unlike other species, ants communicate using a chemical called pheromone which is left on the ground paths in order for the other ants to follow. Some species such as Argentine ant *Iridomyrmex humilis* (Goss et al., 1989) use these pheromones to mark paths from food sources to the nest. An experiment called “Double Bridge” run by Deneubourg and colleagues (Deneubourg et al., 1990; Goss et al., 1989) has shown that in controlled environments, the foraging ants are able to find the shortest path from the food source to the nest by marking them with pheromones as a way of communication.

This experiment demonstrates that the foraging ants have the capacity to modify the environment take decisions based on probabilistic rules in order to find the optimum solution for a problem. It also leads us to the conclusion that this behaviour can be replicated for other optimization problems using artificial ants.

While for the real ants the pheromone evaporation is not important due to being extremely slow, for the artificial ants this process can make the difference between path convergence and searching for new paths. However, for the implementation of this project the evaporation has been ignored and various pheromone levels have been use instead to differentiate the fitness of the paths (more information is provided in the Technical Development section).

3.3.2 The ACO Metaheuristic

The starting point of the ACO Metaheuristic is the behaviour of the real foraging ants and it can be used to solve any combinatorial optimization problems. At a very basic level, a combinatorial problem involves a finite set of feasible solutions each having an associated cost with the aim to find the one with the minimal cost / shortest path. Any combinatorial problem normally becomes the shortest path problem and can be also represented in a graph as shown in Figure 6 below. The graph contains:

- **nodes** representing candidate solutions
- **arcs** indicating the link / relationship between solutions; an arc also suggests that the pair of components is ordered and therefore feasible
- **weight** normally representing the cost / length of the paths between solutions

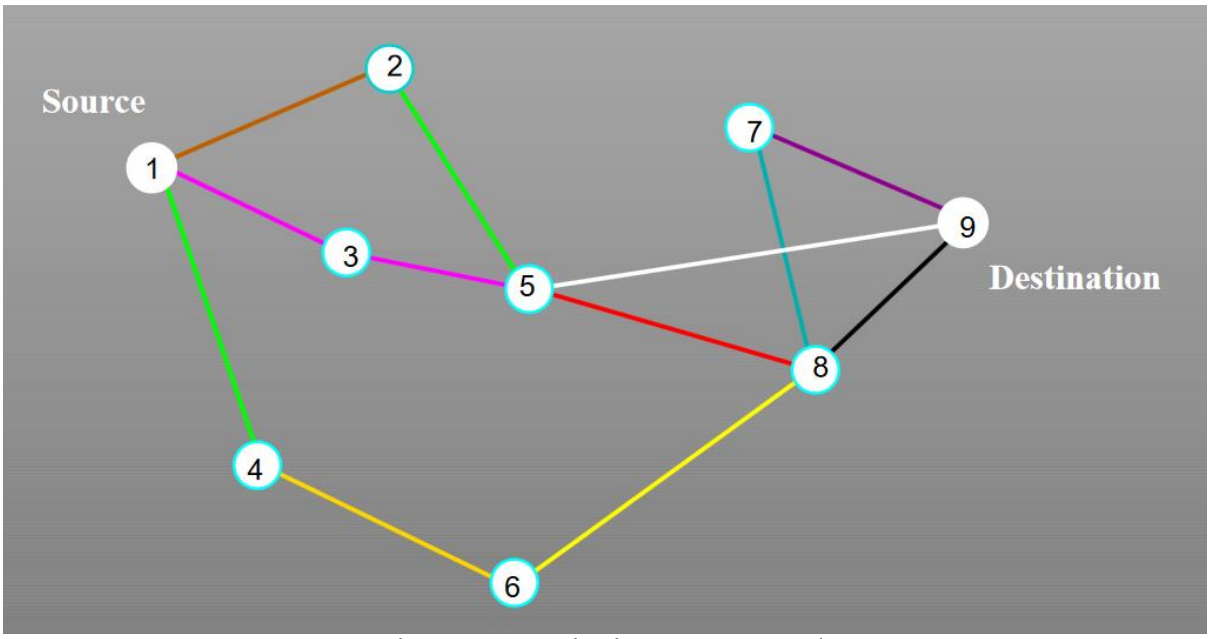


Figure 6 – Example of a component graph

ACO is based on the following principles:

- Ants collectively explore the solution space
- Pheromone trails are left on the paths found to lead to the source of food
- Ants follow pheromone trails laid by other ants
- Pheromone levels of the paths decrease due the chemical substance evaporating as the time passes.

3.3.3 The Metaheuristic – general architecture

The ACO metaheuristic behaviour can be illustrated in pseudo-code based on (Dorigo, M., 2004):

```
procedure ACO_Metaheuristic()
    while (¬stopping_condition)
        Schedule_Activities
            GenerateAntsAndActivity()
            UpdatePheromones()
            DaemonActions()        %optional
        end Schedule_Activities
    end while
return global_best_solution
```

Figure 7 – ACO metaheuristic pseudo-code

ACO metaheuristic is based on three main procedures – GenerateAntsAndActivity, UpdatePheromones and DaemonActions.

GenerateAntsAndActivity manages the ant colony used to explore the space concurrently and find optimal solutions. Ants move from one node to another by making decisions based on the pheromones intensity of the paths and heuristic information. Solutions for the optimization problem are built incrementally and evaluated by the ant in the UpdatePheromones procedure upon which appropriate pheromone amounts are laid.

UpdatePheromones represent the process by which pheromone levels are changed. The intensity of the pheromone trails can either increase when the ant deposits the substance or decrease as the time passes (evaporation). This way the probability or chances for another ant to follow the same path go lower or higher and any adjustments at this point will have an impact on the effective exploration of the space. It is very important to find a way to distribute the pheromones so that the ant's decision is balanced when encountering this situation in order to avoid convergence onto the shortest path.

Because ants can only make an impact locally, DaemonActions allows for any global refinements and adjustments to the search process. A DaemonActions example can be the activation of a procedure meant to change the information used by the ants in the decision making process or a local optimization procedure. Upon a close look at a path, the daemon for instance can permit the ants to deposit a higher or lower amount of pheromones or change the pheromone distribution rules.

The above process is normally run using several iterations depending on the optimization problem. The stopping_condition can be defined as a certain number of iterations which are reached or the solution found is satisfactory enough not to continue the search.

Because ACO is a metaheuristic, the Schedule_Activities construct does not specify how to implement or schedule these three procedures. It is up to the designer to specify how the activities should interact, whether they should be executed in parallel or independently and so on. Although ACO does not provide such a standard, the previous implementations of this algorithm such as the Knapsack Problem have brought recommendations and good practices which can be followed in related optimization problems.

3.3.4 Ant’s decision and convergence problem

The ants build solutions to the problem concurrently based on probability. Since ACO is a probabilistic algorithm the space is not known or defined already but instead it is incrementally discovered by ants based on hints provided by the pheromones. Ants start exploring new solutions based on the pheromone intensity on the paths.

This is not a straightforward problem, in fact the decision of choosing the path with a higher level of pheromones is complex and more considerations are needed when making this choice because the principles of this algorithm tell the ants to follow paths with pheromones but, in the same time, the ants also need to explore unknown space which might lead to new global best solutions.

3.3.5 Knapsack Problem

There are already well-known and successful implementations of the algorithm in problems such as The Traveling Salesman Person (Dorigo & Gambardella, 1997) where Ant Colony is looking for the shortest possible path the salesman needs to follow in order to visit the whole given list of cities only once and return to the origin city.

However, a more relevant implementation of the algorithm is the 0-1 Knapsack Problem which consists of loading objects in to a knapsack in such a way that the obtained total profit of all objects included in the knapsack is maximum and the sum of the weights of all packed objects does not exceed the total knapsack load capacity (Schiff, 2013). The 0-1 part refers to the decision of whether an object can be loaded into the knapsack or not.

The Knapsack problem is one of the NP-Hard (NP: non-polynomial) problems (Garey & Johnson, 1979). It means that the optimal solution is hard to find and it takes huge amounts of time.

At each step, the ant can choose where to move further from a set of feasible solutions called “neighbourhood”. The entire solution for the Knapsack problem is made of different partial solutions found at each step and added as objects to a list. Initially, the set is empty but it is gradually filled with objects chosen based on probability as the ants go from one state to another.

In solving this problem, three algorithms AKA1, AKA2 and AKA3 (Schiff, 2013) were used and their efficiency analysed. In the experiments, various parameters such as number of cycles, evaporation rate and number of ants were either decreased or increased and results were recorded each time. However, it is important to note that one of the algorithms was able to find the maximal profit after just 200 cycles showing a huge potential for the ACO metaheuristic in this kind of combinatorial problems.

4 Technical Development

4.1 Framework

The implementation of the solution described below is based on an already developed framework for DAL allocations used in (Sorokos, et al., 2015) for the previous implementation of Tabu Search Algorithm. This also allows for an easier and better interaction at the level of the code making the implementation more efficient. However, interfacing with this framework involved an Integrated Development Environment and programming language constraint hence why the solution is written in C++ using Visual Studio IDE.

4.2 System Design – Adaptation of ACO for the Aerospace Safety Requirements Allocation problem

The DALs implementation of the algorithm is very similar to the one used in the Knapsack Problem and has been mainly based on the procedures and rules mentioned in the Background section. However, it has been slightly deviated from its original principles and adapted to the DALs allocation problem.

4.2.1 Search Space

A system with N components can be represented in a space with N dimensions where the allocations (solutions) are represented as nodes whose coordinates are the DALs assigned to each component. We can have several subsets (minimum cut sets) which can be explored from the very beginning but not the combinations between solutions within subsets. This means we can sort each of the solutions by DAL cost within each subset according to the distance between the nodes in ascending order.

It leads to the conclusion that once we are in a specific position in the graph we have N choices multiplied by the number of options for each component. We can then simply check the candidate options and choose the one with a level of pheromones which is above a set target or otherwise a random one (whichever is closer) if no pheromones exist (i.e. first iteration).

The stopping criteria can be a certain number of iterations or a time limit without improving the result or if some lower (upper) bound of the result is known and the achieved result is close enough to this bound. The number of dimensions of the graph is equal to the number of components only if we have 1 subset; otherwise the number of dimensions of the graph will be equal to the number of subsets.

4.2.2 Pheromones

For simplicity, it has been decided that in this particular implementation the pheromones do not evaporate / dissipate but instead the pheromone levels do depend on the solution quality. Therefore the focus was more on how the distribution of the pheromones can be done properly so that the convergence is avoided while optimal solutions are effectively found.

The ants only leave pheromones when a new 'best solution' is found. The probabilistic 'next path' choice is random based on the level of pheromones on the path.

4.2.3 Procedures

The basic flow used in implementing the solution (also illustrated in Figure 8):

a) Suballocation packs are retrieved; a suballocation is a partial allocation of DALs to a subset of the model's BasicEvents; a model contains all of the inputted Components (architectural component, part of the model), BasicEvents (basic events resulted from the Fault Tree Analysis), Effects (fault tree top level failures) and CutSets (of fault trees). It allows loading them from file, access to them and allocating DALs to them.

b) Ants are set up – the procedure requires a data structure to keep track of current coordinates of the ants; these ants are then stored in a list.

c) Pheromones are set up – again, it involves a data structure to represent the latest transitions the ants have gone through on a map and keep counters with the level of pheromones on the paths.

d) The first ant is allocated randomly on a solution representing an option from each subset.

e) The ant is then moved to the next option until it finds a better solution than its best or until it reaches an arbitrary number of steps.

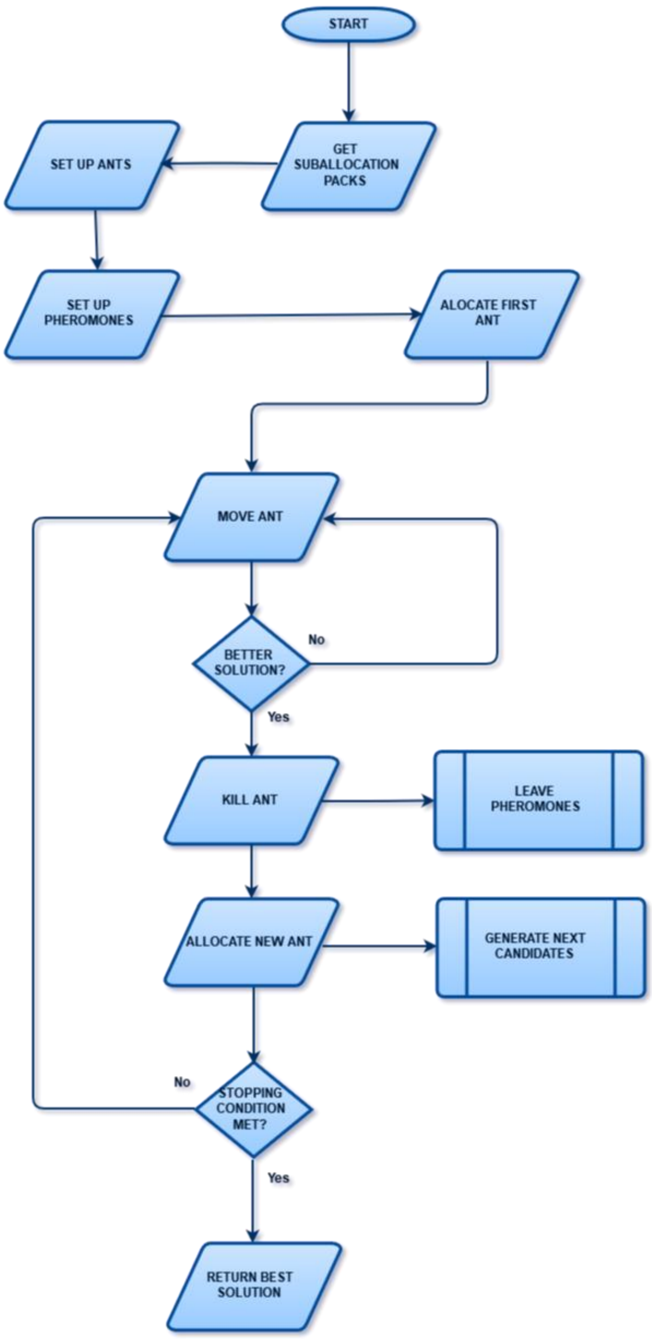


Figure 8 – Flowchart of the implementation mechanisms

f) Ant is then 'killed' and pheromones are left onto the paths the ant has gone through if the path has found better solutions in the way that the paths closer to the solution will contain more pheromones than the others (i.e. max, max-1, max-2, etc).

g) Another ant is then generated and it is allocated on paths that have either higher level of pheromones or some pheromones, otherwise randomly.

h) Next candidates are generated by either incrementing or decrementing the option from the randomly picket subset.

i) The e) to h) procedures are repeated until an arbitrary point in time.

j) Finally, the best solution found is returned.

4.3 Test design and system testing

The main tests have been carried out on an aircraft Wheel Braking System (WBS) which has also been used in assessing the efficacy of Tabu Search in (Sorokos, et al., 2015). The tests aimed to check the results returned by the algorithm against the expected results for the model used. In this particular situation it was important to understand the very best solution that can be achieved (i.e. a cost of 470 was the minimum possible for the WBS model). However, in order to achieve good results, tests have been done at a more granular level and some of these are:

- Model tests – in order to make sure the model loads properly from the framework and to take into account any constraints.

- Ant tests – in order to monitor the ant search space and check whether there are any coordinates that fall outside the feasible solutions space; tests were also run on the ant's latest transitions list so that any patterns in the ant behaviour can be noted and further actions to be taken accordingly.

- Iteration level tests – because this implementation requires several iterations it was extremely important to understand how the search space is created and modified over time; therefore, tests have been run on different variables inside these iterations and any anomalies discovered and fixed.

- Pheromone levels – some of the most useful tests were those run on the pheromone levels of the paths so that they can be adjusted in the DaemonActions procedure mentioned in the Background section; for instance, tests showed that paths had too low pheromone levels which prevented the ants from taking the right decisions; as a result, some adjustments were made in the distribution of the pheromones and levels were inflated using various rationales.

- Best solution tests – were run on the best solution retrieved by the program to check whether it is suitable for the model and its coordinates are found in the allowed search space.

4.4 UI Design

The implementation does not have a User Interface, instead it makes use of the HiP-HOPS' UI. The solution is implemented in the framework provided which communicates with HiP-HOPS through XML and displays the results in its user interface found in the Appendix A.

4.5 System Implementation

4.5.1 Initial Setup

- **Data Structures**
  The program focuses on two main data structures – Ant and Transition (or Path). A transition has a beginning and an end, a choiceIndex of type integer to refer to the index of the component in the list of the model components and pheromoneLvl of type integer as well which contains the pheromone level of the path (initially it is zero which means there are no pheromones laid).

An Ant has coordinates in the search space representing the current position of the ant. These are updated as it moves from one solution to another. It also contains of list of transitions it has made called antTransitions and a cost variable representing the cost of the current ant position.

- **Initializations**
  The implementation relies on many objects already created in the framework and described in the previous sections. It all starts from the system model which is loaded into the program. From this we get the AllocationPacks which is a pool of allocation packs. An allocation pack represents a group of possible partial allocations that were generated by selecting different combinations within the same cut set and taking the same option within the DAL decomposition rules.

  The model also retrieves the number of components which defines the space dimension of the solutions. Therefore, the best solution coordinates (bestSolCoord) will be within this boundary. Initially, it is assumed that the globalBest cost is maximum (50 * numberOfComponens which is 50x13=650). Furthermore, a list called latestTransitions keeps track of all the paths transited by the ants.

  The first ant is then allocated at a random position within the space limits. All the elements from its data structure are updated accordingly to reflect the current state including coordinates and the SubAllocations for each of these.

4.5.2 Repetitive procedures

For a specific number of times the ant is moved and updated in a similar way as the first ant until a better globalBest solution is found. By default, the ant is moved randomly but at that point one of the transitions in the latestTransitions may continue the path the ant is on in which case the ant can follow the pheromone trail and make a transition to that path with a probability based on the pheromone level or stick to the random move.

The ant has random targets when comparing the pheromone levels. They are represented as percentages from 1 to 100 which are checked against the percentage of pheromone levels on the path. If the path has pheromones of a satisfactory intensity for the ant’s preference at that time then it is very likely that the path will be followed by the ant.

Regardless the outcome, the new Allocation and cost of the current ant position are recalculated and the newCost is then compared with the globalBest. If the newCost beats it (it is less than the globalBest) then the globalBest becomes the newCost, the bestSolCoord vector is updated and the stopping criteria is met. The code can also break the loop if the ant has been through 120 (arbitrary number) transitions without finding a new globalBest.

4.5.3 Pheromones Distribution – Arithmetic Progression, Geometric Progression & Exponential Growth

Certainly the most important part which can influence the ant’s decisions and therefore the efficacy of the algorithm is the appropriate distribution of the pheromones onto the paths. If an ant has found a better solution, it will want to leave pheromones on all antTransitions which lead to the solution. It can either be a totally new path or can already by found into the latestTransitions in which case the pheromones are added on top of the existing ones.

The ACO principles suggest that we need to have more pheromones onto paths closer to the solution. However, it is not specified what is the ideal quantity that needs to be laid. For this implementation, a number of ways to force the pheromone levels grow as the paths get closer to the solution:

- **Arithmetic Progression**
  An arithmetic progression is a sequence of numbers such that the difference of any two successive members is a constant. The sum **S** of the first **n** numbers in an arithmetic progression is given by the following formula:

  $$S = \frac{(2a_1 + d(n-1)) \cdot n}{2}$$

  where a1 is the first number and d is the difference between numbers.
  As these numbers are found in a progression, it is sufficient to find a1. Therefore:

  $$a1 = \frac{2S - n \cdot d(n-1)}{2n}$$

  In the implementation, S = costRatio, d is an arbitrary difference chosen and n is the size of the antTransitions. The costRatio is the total amount of pheromones to be distributed and is calculated as:

  $$Cost\ Difference = \frac{allocation\ cost\ at\ start - allocation\ cost\ at\ the\ end\ of\ path}{allocation\ cost\ at\ start}$$

  By replacing these values, the final equation will be:

  $$a1 = \frac{2 \cdot costRatio - antTransitions \cdot d(antTransitions - 1)}{2 \cdot antTransitions}$$

  If we rename a1 with previousTerm, the first path will have pheromoneLvl = previousTerm, with previousTerm becoming previousTerm + d and being assigned to the next path and so forth.

- **Geometric Progression**
  With the fast growth of pheromone levels in mind another solution is the geometric progression. In mathematics, a geometric progression (also inaccurately known as a geometric series) is a sequence of numbers such that the quotient of any two successive members of the sequence is a constant called the common ratio of the sequence. The sum **Sn** of the first **n** numbers in a geometric progression is:

  $$Sn = a1 \cdot \frac{1 - q^n}{1 - q}$$

  where a1 is the first number and q is the common ratio.
  In this case, we want to find a1 so that from there each next term will be based on the previous one using the formula for the ***n-th*** term:

$$a_n = a_{n-1} \cdot q$$

Therefore:

$$a1 = \frac{Sn \cdot (1-q)}{1-q^n}$$

In the implementation, Sn = costRatio, q is an arbitrary factor chosen and n is the size of the antTransitions.

Due to very low pheromones, the costRatio has been arbitrarily multiplied by 20 to inflate the levels.

By replacing the third formula we obtain:

$$a1 = \frac{costRatio \cdot (1-q)}{1-q^{antTransitions.size()}}$$

If we rename a1 with previousTerm, the first path will have pheromoneLvl = previousTerm, with previousTerm becoming previousTerm * q (using the second formula) and being assigned to the next path and so forth.

- **Exponential Growth**
  In order for the pheromones to grow even faster, there are several approaches based on the exponential growth. A related formula which illustrates such a growth is:
  $$f(x) = \frac{1}{e^x}$$
  where x is the path index which is incremented with time.
  However, this would lead to a fast decrease in levels rather than increase as shown in the Figure 9 below:

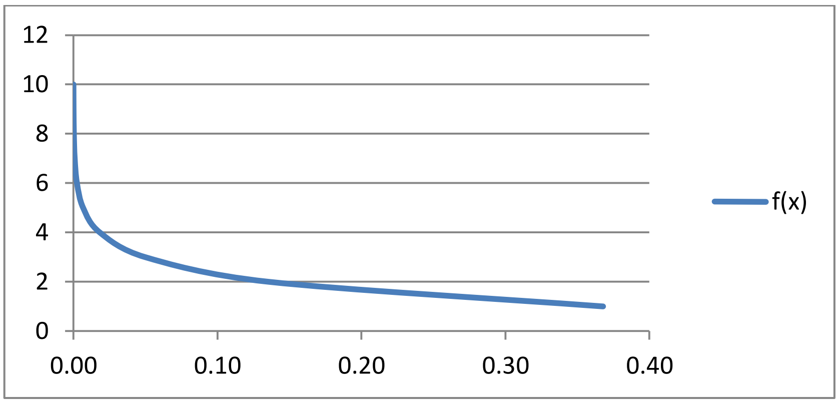


Figure 9 – Example of decrease using Exponential Growth

In order to force the levels go up quickly, the formula has been adapted as follows:

$$f(x) = 1 - \frac{1}{e^x}$$

This way it is possible to inverse the graph and obtain the desired result in Figure 10 below:

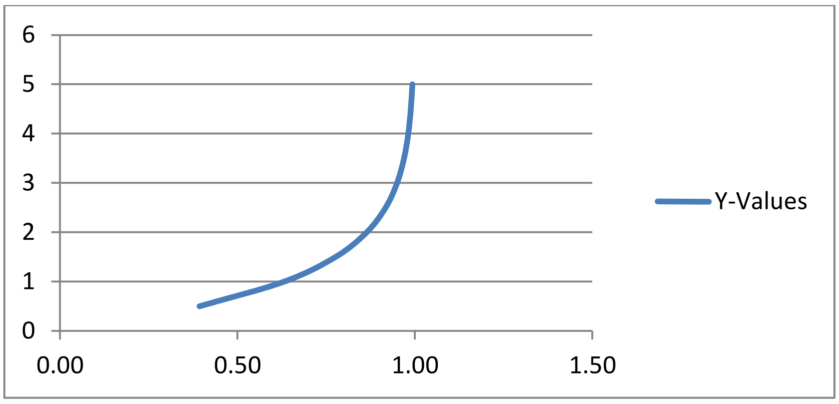


Figure 10 – Example of increase using Exponential Growth

In this graph the value of the pheromones would only be between 0 and 1. This limit can be conveniently replaced with costRatio thus resulting in the following formula:

$$f(x) = costRatio - \frac{costRatio}{e^{pathIndex}}$$

The previousTerm starts at 0 and is allocated to the pheromoneLvl of the first path. Then, the above formula is applied and previousTerm becomes costRatio – (costRatio/(e^pathIndex)) with pathIndex starting at 1. Finally, pathIndex is incremented and the procedure is repeated.

5 Evaluation

Several runs of the program have been made in order to assess the performance of various permutations of the ACO algorithm. These have been executed on an Intel®Core™ i5-4570 CPU @ 3.20Ghz machine. In the program there were 1000 iterations (ants) executed for each run which took 5 seconds in average. The methods assessed (with 100 runs each) consisted of different ways of distributing the pheromone levels on the paths with a low factor (replacing q or d in the previous formulas with 5), high factor (10), low inflation of pheromone levels (costRatio is multiplied by 2) and high inflation (costRatio is multiplied by 20):

1) Arithmetic Progression with low factor and low inflation (APLL)
2) Arithmetic Progression with low factor and high inflation (APLH)
3) Arithmetic Progression with high factor and high inflation (APHH)
4) Arithmetic Progression with high factor and low inflation (APHL)
5) Geometric Progression with low factor and low inflation (GPLL)
6) Geometric Progression with low factor and high inflation (GP
7) Geometric Progression with high factor and high inflation
8) Geometric Progression with high factor and low inflation
9) Exponential Growth with low inflation
10) Exponential Growth with high inflation

In assessing the performance of the above methods a number of criteria have been considered:

a) Number of occurrences (470) – how many times the best solution (470) was found (out of 100 possible times) – Figure 11
b) Number of occurrences (490) – how many times the second best solution (490) was found (out of 100 possible times) – Figure 12
c) Avg Number of Iterations (470) – how many iterations in average were needed to find the best solution – Figure 13
d) Avg Number of Iterations (490) – how many iterations in average were needed to find the second best solution – Figure 14
e) Avg Time in seconds (470) – how long it took in average to find the best solution – Figure 15
f) Avg Time in seconds (490) – how long it took in average to find the seconds best solution – Figure 16
g) Total Time in seconds (100 runs) – the total time it took to make 100 runs (of 1000 iterations/ants each) – Figure 17

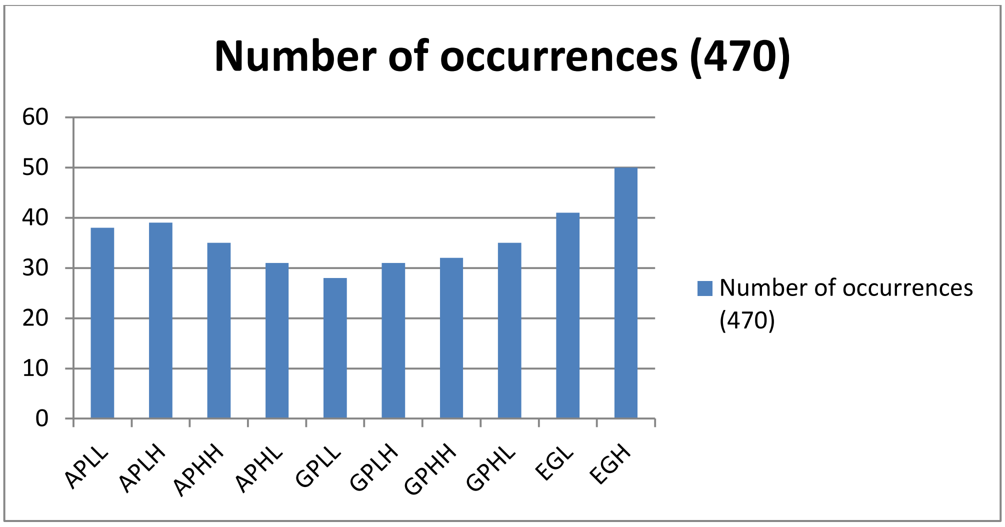


Figure 11 – Number of occurrences (470)

Certainly, the most important criteria is the number of times the best solution was retrieved. Figure 11 shows that the Exponential Growth formula has found the best solution in 50 cases out of 100 which makes it the most efficient method while the Geometric Progressions prove to be less efficient.

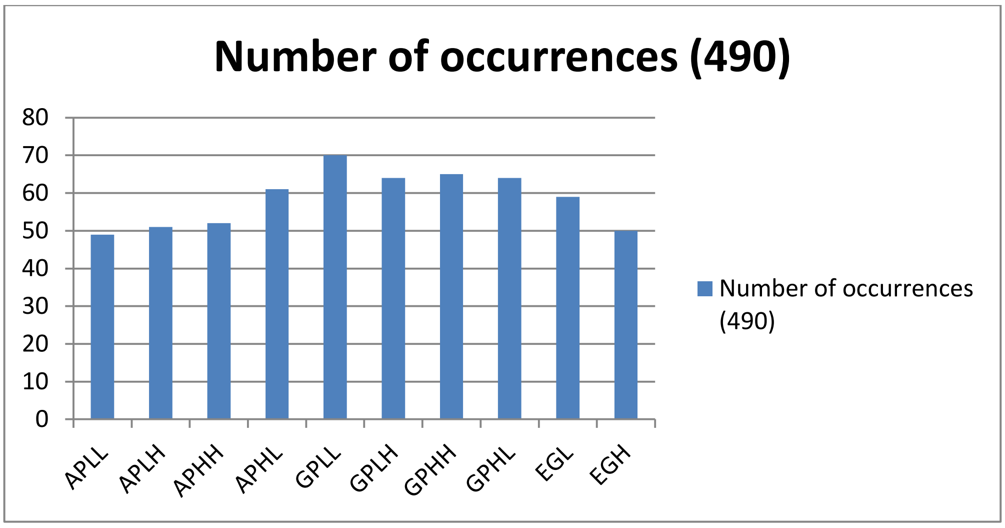


Figure 12 – Number of occurrences (490)

Although finding the best solution is the most important, it is worth assessing the performance for finding the second best solution as well. Figure 12 shows that the Geometric Progressions perform better with regards to this matter while Arithmetic Progressions are less efficient in this case.

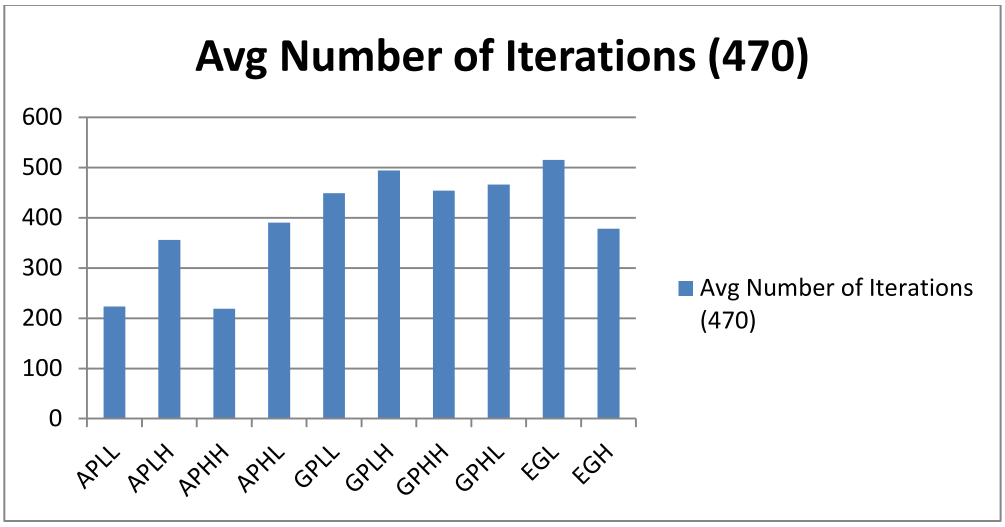


Figure 13 – Average Number of Iterations (470)

Figure 13 shows the number of ants needed to find the best solution and it seems that Arithmetic Progressions are relatively efficient with a bit over 200 ants needed to find 470.

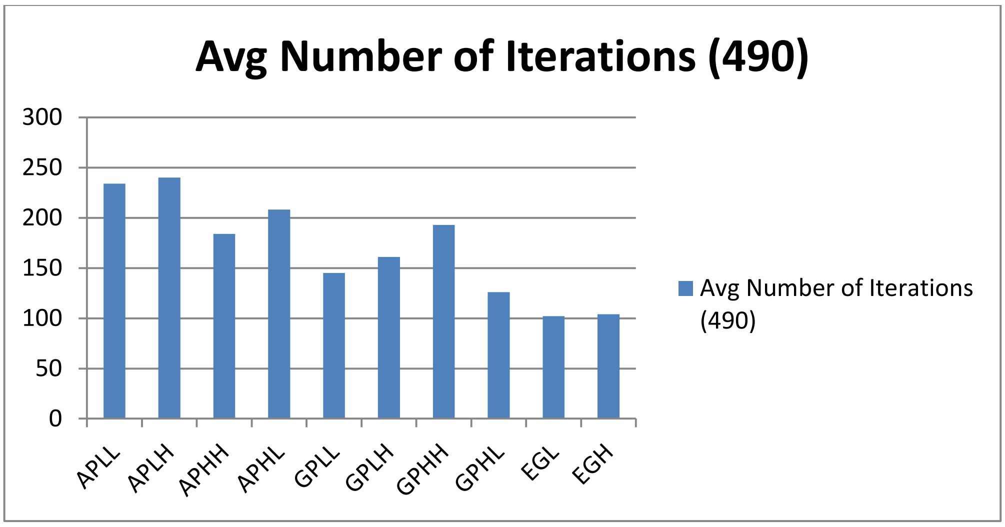


Figure 14 – Average Number of Iterations (490)

It is also worth mentioning that although the best solution may require more ants, the second best solution is generally found after just 100 iterations with the Exponential Growths. In this situation the Arithmetic Progressions are the least performant and therefore are not recommended.

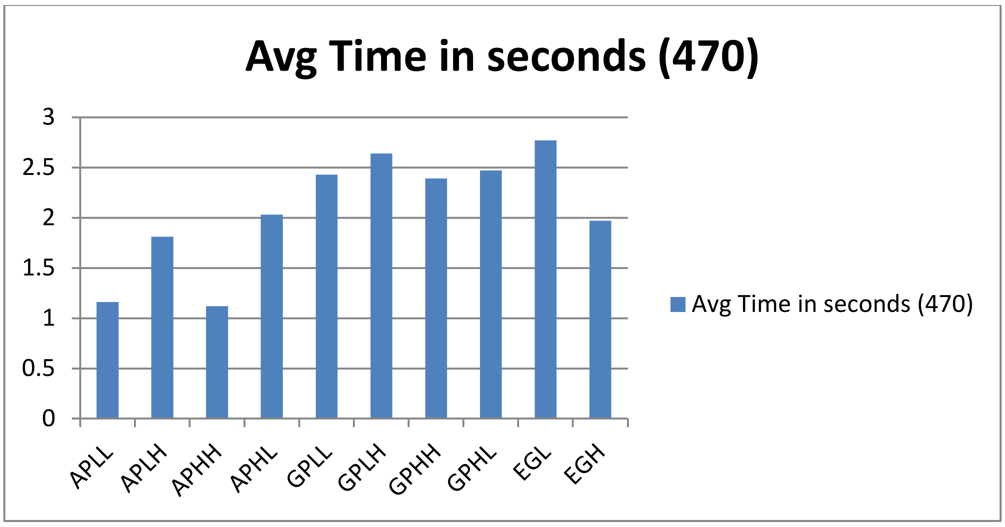


Figure 15 – Average time in seconds (470)

The best solution was found in a bit over 1 second using the Arithmetic Algorithms with low factor / low inflation and high factor / high inflation combinations. The most time taken to reach the best solution is almost 3 seconds recorded by the Exponential Growth with low inflation.

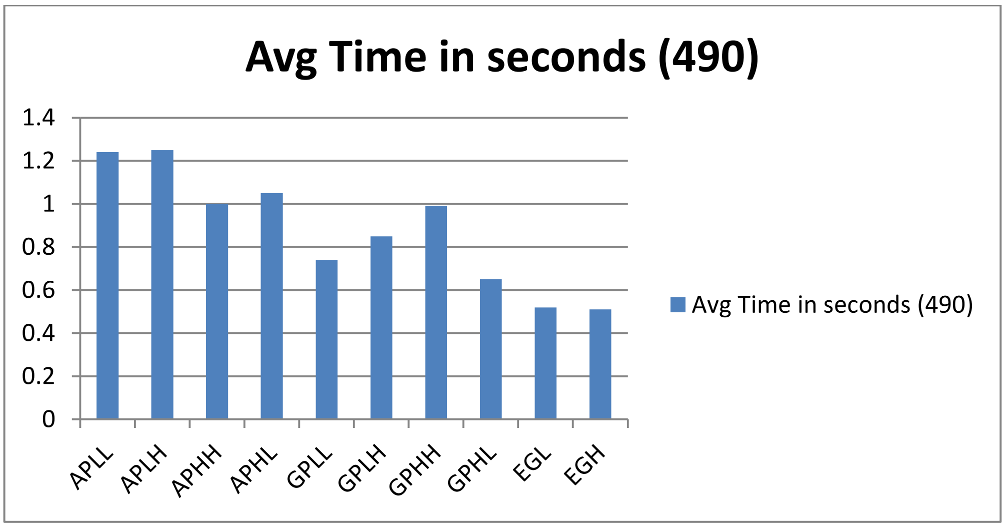


Figure 16 – Average time in seconds (490)

Although Exponential Growths are worst when searching for the best solution, they are excelling at finding the second best solution quickly. In fact, it only took half a second for these formulas to find the second optimal solution.

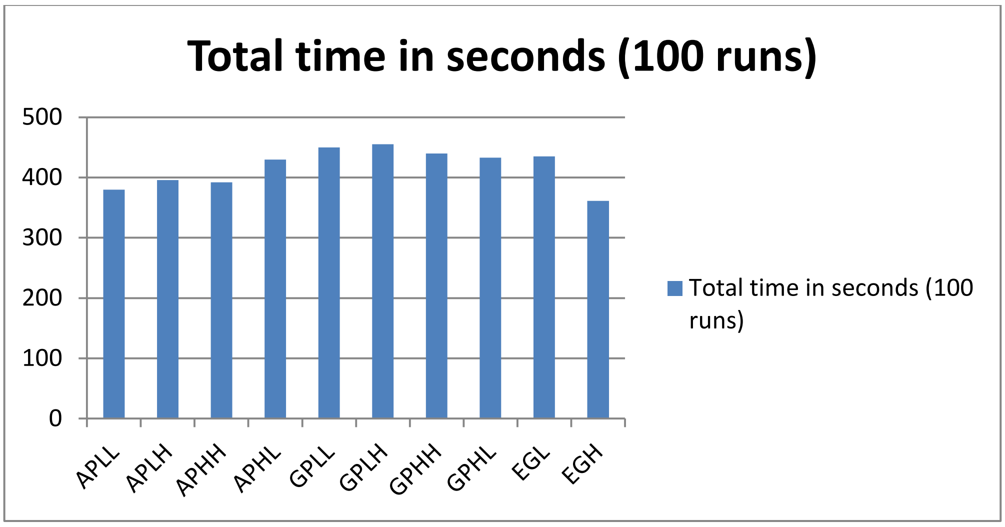


Figure 17 – Total time in seconds (100 runs)

Finally, the time required for running each method 100 times was also recorded. With minor variations, the program executed all formulas in in around 400 seconds each.

Table 2 below shows the performance summary showing the strengths and weaknesses of each method used:

Method	Number of occurrences (470)	Number of occurrences (490)	Avg No of Iterations (470)	Avg No Iterations (490)	Avg Time in seconds (470)	Avg Time in seconds (490)	Total time in seconds (100 runs)
APLL		Worst					
APLH				Worst		Worst	
APHH			Best		Best		
APHL							
GPLL	Best	Best					
GPLH							Worst
GPHH							
GPHL							
EGL			Worst	Best	Worst		
EGH	Worst					Best	Best

Table 2 – Performance summary (strengths and weaknesses)

To sum up, there is no best formula for every criteria but it is worth mentioning that the Geometric Progression with low factor and low inflation is the best for finding the most optimal solution in half of the cases while the Arithmetic Progression with high factor and high inflation is best at finding the best solution in the shortest possible time. Further comparisons between these approaches can be found in Appendix A.

5.1 Further Work

Due to the nature of the metaheuristic, ACO was not able to find the best solution at every run and it proved to be less efficient than the Tabu Search algorithm. However, the algorithm was able to retrieve either the first or the second best solution in more than 90% of the cases which classifies the ACO not necessarily as a fast algorithm but as a relatively reliable one.

At the beginning of the project, there were more metaheuristics planned to be implemented (Genetic Algorithms, Simulated Annealing, etc). Due to time constraints, it was only reduced to one but with different permutations to be applied. In the end, these consisted of various ways of distributing the pheromone levels which were particularly challenging because of the different combinations of variables that needed to be adjusted in order to achieve optimal results. If more time was available, the ants could have been forced to run away from the best solutions and explore new paths rather than following the same ones.

Certainly this ACO algorithm can be improved by finding other ways of distributing the pheromones. It would be interesting to see how the algorithm behaves when adjusting the factor and the inflation level either higher or lower and running the program after each try.

Multithreading is also important to be considered especially for obtaining the optimal solutions faster. This would however require more attention to the common resources used by the ants such as the antTransitions and making sure that these are appropriately locked while they are updated by the other ants.

6 Conclusion

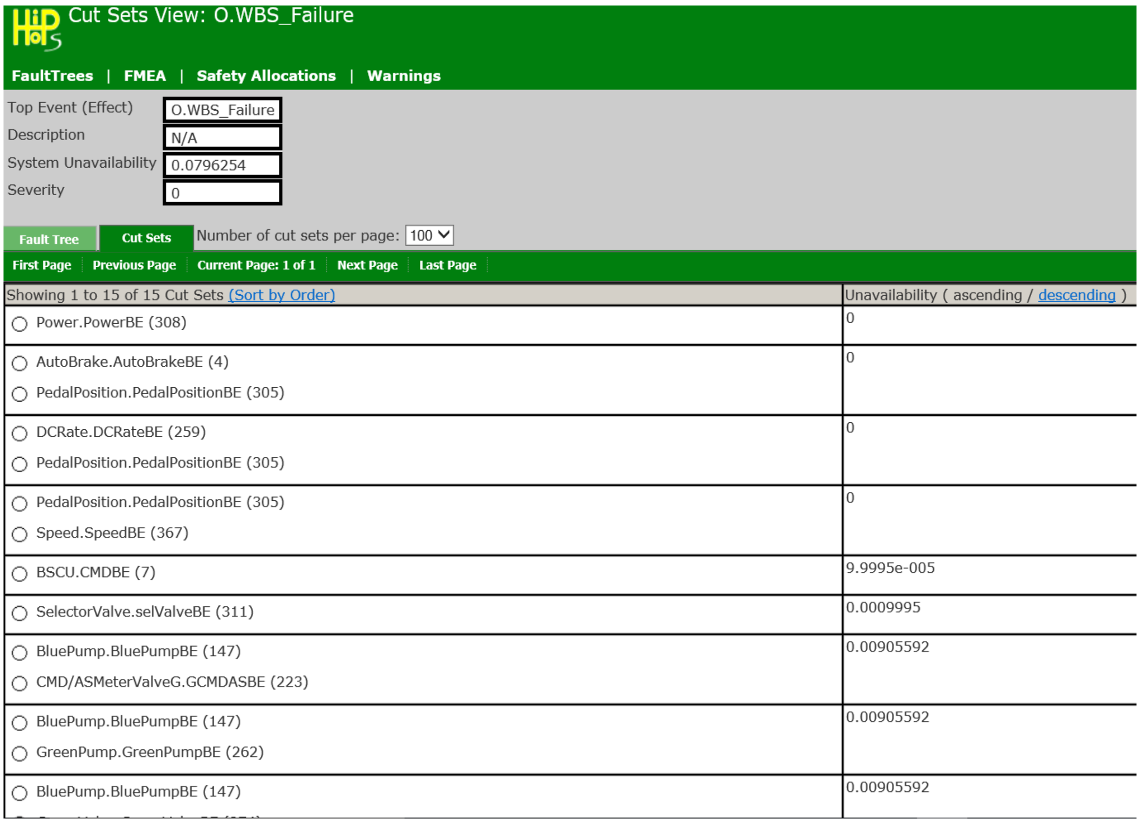
This project has managed to automate the allocation of safety requirements in the Aerospace Industry through the exploration of possible improvements via application of ACO metaheuristic and parallel exploration of the search space in many places. It has also contributed to the state-of-the-art research that aims to optimize the process of assigning safety requirements in complex systems within the aerospace industry.

The project has addressed one of the main issues in the Aerospace Industry which is the manual allocation of safety requirements. Because of the very high scale of the aircraft systems nowadays, this task has become infeasible and very time consuming if executed manually.

This project has looked into ways of automating this process efficiently and reliably by using a metaheuristic which has not been previously adopted in the DALs allocation problem. Previous work was analysed in order to understand the different approaches to this matter. Ant Colony Algorithm was picked based on its proven efficiency in other similar problems such as the Knapsack Problem.

In conclusion, the ACO algorithm is far from being fully explored but there is a lot of potential brought by this algorithm and further improvements can be made in order to optimize it.

Appendix A: ACO Results in HiP-HOPS User Interface

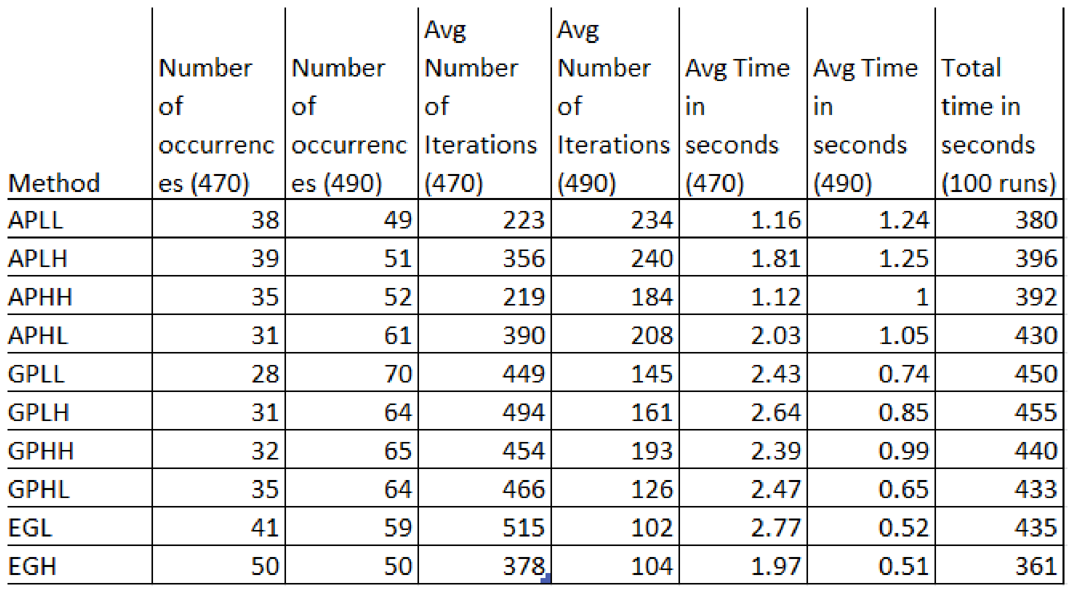


Top Event (Effect)	O.WBS_Failure
Description	N/A
System Unavailability	0.0796254
Severity	0

Number of cut sets per page: 100

Showing 1 to 15 of 15 Cut Sets (Sort by Order)	Unavailability (ascending / descending)
○ Power.PowerBE (308)	0
○ AutoBrake.AutoBrakeBE (4) ○ PedalPosition.PedalPositionBE (305)	0
○ DCRate.DCRateBE (259) ○ PedalPosition.PedalPositionBE (305)	0
○ PedalPosition.PedalPositionBE (305) ○ Speed.SpeedBE (367)	0
○ BSCU.CMDBE (7)	9.9995e-005
○ SelectorValve.selValveBE (311)	0.0009995
○ BluePump.BluePumpBE (147) ○ CMD/ASMeterValveG.GCMDASBE (223)	0.00905592
○ BluePump.BluePumpBE (147) ○ GreenPump.GreenPumpBE (262)	0.00905592
○ BluePump.BluePumpBE (147)	0.00905592

Appendix B: Full comparison between the methods used in the pheromone distribution



Method	Number of occurrenc es (470)	Number of occurrenc es (490)	Avg Number of Iterations (470)	Avg Number of Iterations (490)	Avg Time in seconds (470)	Avg Time in seconds (490)	Total time in seconds (100 runs)
APLL	38	49	223	234	1.16	1.24	380
APLH	39	51	356	240	1.81	1.25	396
APHH	35	52	219	184	1.12	1	392
APHL	31	61	390	208	2.03	1.05	430
GPLL	28	70	449	145	2.43	0.74	450
GPLH	31	64	494	161	2.64	0.85	455
GPHH	32	65	454	193	2.39	0.99	440
GPHL	35	64	466	126	2.47	0.65	433
EGL	41	59	515	102	2.77	0.52	435
EGH	50	50	378	104	1.97	0.51	361

References

AircraftEngineering. 2016. *AircraftEngineering | AIRCRAFT PASSIONATES*. [ONLINE]
Available at: https://aircraftengineering.wordpress.com/.
[Accessed 30 April 2016].

Bieber, P., Delmas, R. & Seguin, C., 2011. DALculus - Theory and Tool for Development Assurance Level Allocation. Naples, Italy, Springer Berlin Heidelberg, pp. 43-56.

Coit, D.W. & Smith, A.E., 1996. Reliability optimization of series-parallel systems using a genetic algorithm. Reliability, IEEE Transactions on, 45(2), pp.254-260.

Deneubourg, J.L., Aron, S., Goss, S. and Pasteels, J.M., 1990. The self-organizing exploratory pattern of the argentine ant. Journal of insect behavior, 3(2), pp.159-168.

Dorigo M., 2004. *Ant Colony Optimization (MIT Press)*. First Edition, First Printing Edition. A Bradford Book.

Dorigo, M. & Gambardella, L.M., 1997. Ant colonies for the travelling salesman problem. BioSystems, 43(2), pp.73-81.

Hobbs, C., 2016. *Applying Bayesian belief networks to fault tree analysis of safety critical software | Embedded*. [ONLINE]
Available at: http://www.embedded.com/design/prototyping-and-development/4206019/Bayesian-Fault-Tree-Analysis-of-Safety-Critical-Software.
[Accessed 30 April 2016].

Engineering New World | Is your Complex System Project on track for Ultraquality Implementation?. 2016. Engineering New World | Is your Complex System Project on track for Ultraquality Implementation?. [ONLINE]
Available at: http://www.engineeringnewworld.com/?p=257.
[Accessed 30 April 2016].

Garey, M.R. & Johnson, D.S., 1979. A Guide to the Theory of NP-Completeness. WH Freemann, New York.

Glover, F., 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, pp. 533-549.

Goss, S., Aron, S., Deneubourg, J.L. and Pasteels, J.M., 1989. Self-organized shortcuts in the Argentine ant. Naturwissenschaften, 76(12), pp.579-581.

Holland, J.H., 1975. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press.

Maragakis, I., Clark, E., Piers, N. and Prior, N.D., 2009. GUIDANCE ON HAZARDS IDENTIFICATION.

Papadopoulos, Y. et al., 2011. Engineering failure analysis and design optimisation with HiP-HOPS. Engineering Failure Analysis, pp. 590-608.

Papadopoulos, Y., 2015. Automating Allocation of Development Assurance Levels: an extension to HiP-HOPS. [PowerPoint presentation]. *Automating Allocation of Development Assurance Levels: an extension to HiP-HOPS.* University of Hull, Department of Computer Science, 27 May.

S-18, SAE, 2010. ARP4754-A Guidelines for Development of Civil Aircraft and Systems, s.l.: SAE Int.

Schiff, K., 2013. Ant Colony Optimization algorithm for the 0-1 Knapsack Problem. *Ant Colony Optimization algorithm for the 0-1 Knapsack Problem*. Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology. [ONLINE].
Available at:
https://suw.biblos.pk.edu.pl/resources/i4/i5/i7/i3/i1/r45731/SchiffK_AntColony.pdf
[Accessed 30 April 2016].

Sharvia, S. & Papadopoulos, Y., 2011. IACoB-SA: an Approach towards Integrated Safety Assessment. Trieste, Italy, IEEE, pp. 220-225.

Sorokos, I., Papadopoulos, Y., Azevedo, L., Parker, D. and Walker, M., 2015. Automating Allocation of Development Assurance Levels: an extension to HiP-HOPS. IFAC-PapersOnLine, 48(7), pp.9-14.

Statistical Summary of Commercial Jet Airplane Accidents (2015). [ONLINE]
Available at:
http://www.boeing.com/resources/boeingdotcom/company/about_bca/pdf/statsum.pdf
[Accessed: 27 April 2016].

TC 22/SC3, 2011. ISO 26262 - Road vehicles - Functional safety, s.l.: International Organization for Standardization.

Toksari, M.D., 2016. A hybrid algorithm of Ant Colony Optimization (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption: Case of Turkey. International Journal of Electrical Power & Energy Systems, 78, pp.776-782.

Walker, M.D., 2009. Pandora: a logic for the qualitative analysis of temporal fault trees (Doctoral dissertation, The University of Hull).