AGENT-BASED UNDER HOOD PACKING

Karthik Ravindranath
Clemson University, kravind@clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Part of the Mechanical Engineering Commons

Recommended Citation
Ravindranath, Karthik, "AGENT-BASED UNDER HOOD PACKING" (2011). All Theses. 1049.
https://tigerprints.clemson.edu/all_theses/1049

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
AGENT-BASED UNDER HOOD PACKING

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

by
Karthik Ravindranath
May 2011

Accepted by:
Dr. Georges M. Fadel, Committee Chair
Dr. Lonny L. Thompson
Dr. Gang Li
ABSTRACT

Improving vehicle performance and passenger comfort has been a prime engineering concern and focus of research for many years in automotive design. Turning to high-performance components in an effort to improve vehicle performance alone is often not enough and their placement and interactions with other components should also be an integral part of the improvement process. With the advancement in hybrid electric vehicle technology, the packing of components under the hood is ever more essential and challenging. Under hood packing is a multi-objective optimization problem with many, and mostly conflicting objectives. A non-deterministic multi-objective evolutionary algorithm needs to be integrated with the packing algorithm to obtain solutions. However, it is almost impossible to find optimal solutions in a limited amount of time due to the computationally intensive algorithm. Therefore, a new and efficient approach needs to be developed.

This study applies an agent-based approach to the under hood vehicle packing problem with three objectives, namely: center of gravity, survivability, and maintainability subject to no overlap among components and with the enclosure, and minimum ground clearance. As per the weak notion of agency, a layered architecture is built with an agent on top of object model. A non-deterministic evolutionary multi-objective algorithm (AMGA-2) is used to identify non-dominated solutions, speed up the convergence to a non-dominated set and prevents unpredictability in the agent system. The developed agent-based model is applied to a passenger car but, it can also address
large packing problems for SUVs and Trucks (FMTV). This work demonstrates the
applicability and benefits of an agent-based approach to the packing problem.
DEDICATION

matru devo bhava
pitrudevo bhava
acharyadevo bhava

{Admire your mother, father and teacher like god}

I dedicate this work to my mother Mrs. H.V. Savithri & father Mr. Ravindranath S. P.

Thanks and regards to my sister Dr. Nagashree S. R. & brother-in-law Mr. Manohar S.

One big warm hug to my nephew Rithvik.
ACKNOWLEDGEMENTS

I deeply owe my sincere gratitude to Dr. Georges Fadel for his patience, support and encouragement in completing this thesis and beyond. This thesis wouldn’t have been possible without his guidance and resources. The kind of flexibility and freedom that Dr. Fadel gives makes all the difference. Apart from his knowledge, I am truly inspired by his friendly nature, modesty and positive approach. This is probably the reason he is something more than a superior and mentor.

I would also like to thank my thesis committee members Dr. Lonny Thomson and Dr. Gang Li for their suggestions and guidance. I would specially like to thank the Automotive Research Center for sponsoring this research.

I would like to thank Dr. Sundeep Samson, Dr. Santosh Tiwari and Dr. Paolo Guarneri for their valuable inputs. I would like to express my regards for my fellow CREDO and CEDAR lab mates for their help, coordination and motivation at all times.

Best regards to my friends Niranjan, Sreekanth and Raviteja for always being there.

I would like to express my humble gratitude to all the teachers at Clemson University for the effort they put in to prepare thousands of students like me to take on real-life challenges. Finally, I would like to express my gratefulness to the Department of Mechanical Engineering, Office of International Affairs and Clemson University for providing me an opportunity to study and do research in this excellent academic environment.
# TABLE OF CONTENTS

AGENT-BASED UNDER HOOD PACKING .......................................................... i
ABSTRACT ........................................................................................................ ii
DEDICATION ..................................................................................................... iv
ACKNOWLEDGEMENTS .................................................................................. v
LIST OF TABLES .............................................................................................. viii
LIST OF FIGURES ............................................................................................ ix
INTRODUCTION ............................................................................................... 1
  1.1 Vehicle packing ......................................................................................... 1
    1.1.1 Relevant work .................................................................................... 2
  1.2 Agent-based approach .............................................................................. 5
    1.2.1 History of agents ................................................................................ 5
    1.2.2 Agent definition ................................................................................ 9
    1.2.3 Agent environment .......................................................................... 10
    1.2.4 Relevant work .................................................................................. 11
PROBLEM FORMULATION ............................................................................ 15
  2.1 Constraints ............................................................................................... 15
    2.1.1 Collision detection .......................................................................... 15
    2.1.2 Ground clearance ............................................................................ 23
  2.2 Objectives ................................................................................................. 26
    2.2.1 Center of gravity ............................................................................. 26
    2.2.2 Vehicle maintainability ................................................................... 28
    2.2.3 Vehicle survivability ....................................................................... 31
AGENT-BASED UNDER HOOD PACKING .................................................... 36
  3.1 Assessment of suitable notion of agency ................................................. 36
  3.2 Agent-based architecture ....................................................................... 38
RESULTS ................................................................................................................................. 46

CONCLUSION ......................................................................................................................... 58
  5.1 Contributions .................................................................................................................. 58
  5.2 Future work ...................................................................................................................... 59

REFERENCES ....................................................................................................................... 60
LIST OF TABLES

1.1 Five generations of programming languages ............................................................. 6

2.1 List of components with weights and vertical location of center of gravity ............... 27
2.2 Components and their corresponding maintainability weights .................................... 29

4.1 Packing sequence with degrees of freedom assigned to each component ................. 46
4.2 Objective function values for four examined configurations ........................................ 48
4.3 Under hood packing Java package .............................................................................. 51
LIST OF FIGURES

1.1 Progression of programming language ................................................................. 5
1.2 Agent – A human oriented higher-level software abstraction .................................. 8
1.3 Agent – A software entity ......................................................................................... 10

2.1 Object and its enclosing bounding box .................................................................... 16
2.2 Bounding box evaluation .......................................................................................... 17
2.3 Case where bounding box collision detection test fails .......................................... 18
2.4 Radiator and coolant reservoir tank enclosed in their respective bounding box ...... 19
2.5 Voxel matrix generated for radiator and coolant reservoir tank ............................... 20
2.6 Voxel matrices after facet-box overlap detection ..................................................... 21
2.7 Voxel inversion performed on the voxel matrices .................................................. 22
2.8 Voxels of Engine and radiator models overlapping ................................................. 23
2.9 Three clearance angles used to compute ground clearance ..................................... 24
2.10 Process of a vehicle moving over an uphill ............................................................ 25
2.11 Location of center of gravity from ground .............................................................. 27
2.12 Configuration with components removed along positive X-direction ................... 30
2.13 Configuration with components removed along negative X-direction ................. 31
2.14 Survivability calculation for three components ..................................................... 34
2.15 Under hood vehicle configuration design problem formulation ............................ 34

3.1 Building agents with object-oriented paradigm ....................................................... 39
3.2 Detailed agent-based architecture for the packing problem .................................... 41
3.3 Multi-agent system class diagram .......................................................................... 42

4.1 Survivability vs. Maintainability ............................................................................. 47
4.2 Four non-dominated vehicle configurations selected ............................................ 48
4.3 Four vehicle configurations examined .................................................................... 49
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4 List of non-dominated solutions</td>
<td>50</td>
</tr>
<tr>
<td>4.5 Steps in adding new packing objective</td>
<td>55</td>
</tr>
<tr>
<td>4.6 Steps in adding new component to packing problem</td>
<td>56</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 VEHICLE PACKING

The under hood vehicle configuration design problem also called packing or packaging or under hood vehicle layout design problem is NP-complete [1]. The under hood vehicle configuration design is a complex multi-objective optimization problem. It involves searching optimal non-overlapping locations of components under the hood which result in best or improved vehicle performance subject to some equality or inequality constraints. This work involves three conflicting objectives namely, minimizing center of gravity height, maximizing vehicle maintainability and maximizing survivability. The no overlap between the components and between components and enclosure, and a minimum ground clearance form the two constraints of the problem. The vehicle considered in this work is a Ford Taurus, a full size sedan available in front- or all-wheel drive. Even though, a passenger car is considered, this work can address much larger vehicles like SUVs and Trucks (FMTV).

In the component packing optimization problem, the overlap detection between two or more components and between components and the enclosure is the most expensive operation [2]. To reduce computational cost, the operation can be performed at two levels (coarse and refined) [2]. Initially, a simple and fast collision detection method based on the axis aligned bounding box [3] concept is applied; at the next level, a
fully sophisticated and robust method based on voxel-triangle overlap principle is employed [4, 5]. Configurations comprised of components which overlap or fail the minimum ground clearance angle are rejected to avoid objective functions evaluation. Only the configurations which satisfy both the constraints are selected for objective function evaluation. Minimizing the vertical location of the center of gravity results in configurations which have better vehicle stability and are less susceptible to over-turning. Maximizing maintainability which involves computing the number of components that have to be removed to reach another component results in configurations which are easy to access and therefore maintain. Finally, maximizing survivability yields configurations which are less prone to damage from missile and bullet attacks from the sides, or front of the vehicle.

1.1.1 RELEVANT WORK

Grignon et al. [6] formulates engineering configuration design problem as a multi-objective optimization problem as below:

“Given:

A set of $N$ components defined by their shape, material and position in space
A set of equality and inequality constraints
A set of $M$ objective functions

Find:

A set of design variables $x$ representing a vector of all locations of all components of the system optimizing $F$, a vector of objective function

Satisfying:

Equality and inequality constraints”
Wodziak [7] presents a methodology to solve packing type problems and applies it to the placement of goods in a rectangular volume in order to obtain a desired center of gravity location. The proposed methodology uses a genetic algorithm (GA) to obtain near optimal location of goods for one dimensional, two dimensional, and two and half dimensional packing type problems. The algorithms developed are applied to packing rectangular boxes in a Fruehauf trailer.

Yi et al. [8] discuss that identifying the global optimum using exhaustive search methods in a reasonable amount of time is impossible due to the computationally intensive packing algorithms. This work presents a GA with specialized genome design for packing problems called the packing genome. It uses the GA with a new encoding method and GA operators, and applies it to the vehicle configuration design problem. This NP-complete multi-objective problem considers three conflicting vehicle design objectives: survivability, maintainability and vehicle rollover tendency. The packing GA is integrated with a multi-objective genetic algorithm (NSGA-2) to search for non-dominated solutions. However, solving multi-objective packing problems efficiently is still a challenging task, especially when the problem involves objects with complex geometric shapes.

Gantovnik et al. [9] an extension of Yi et al. [8], apply a multi-objective evolutionary algorithm for solving the configuration design of US Army trucks from the Family of Medium Tactical Vehicles (FMTV). This study considers three objective functions: vehicle dynamic performance, survivability and maintainability subject to overlap detection and ground clearance. Optimization is performed using a combination of the
packing GA and packing sequence. The packing GA is a tailored NSGA-2 with modifications in its encoding and GA operators specifically for the packing problems. The packing sequence describes the order of placement of all components and defines their corresponding relative coordinate systems. The application of relative coordinates prevents the use of additional mechanical and functional constraints and reduces the number of design variables and hence complexity. This work demonstrates that the packing GA outperforms the traditional binary GA for this kind of problems.

Studies in the past [8, 9] show that solving a multi-objective configuration problem is still a challenging task, especially when designs involve components with complex shapes. It is almost impossible to find optimal solutions in limited time due to the computationally intensive packing algorithms. Therefore, a new and efficient approach needs to be developed.

According to Davidsson [10], agent-based approaches are preferable over mathematical optimization techniques when the problem domain is large, the domain is modular in nature and the probability of node or link failure is high. However, mathematical optimization techniques are preferable when finding a system’s optimal or near optimal solution is essential. Thus, the properties of agent-based approaches and classical optimization techniques complement each other. It would be beneficial to combine these two approaches. Davidsson [10] recommends, "Agentify" the optimization algorithms to incorporate some of the features of agent-based approaches.

The following sections cover briefly what agents are and how the agent-based paradigm was selected.
1.2 Agent-based approach

1.2.1 History of Agents

Since 1945, programming languages have advanced significantly. This historic progression has resulted in the development of a number of programming languages written for solving different types of problems and for different types of platforms. This development of programming languages has been mainly categorized into five generations as shown in Figure 1.1 [11]. With each successive generation, programming languages have become more abstract, more user-friendly and more powerful than in the previous generation.

Figure 1.1: Progression of programming language
<table>
<thead>
<tr>
<th>Generation</th>
<th>Period</th>
<th>Examples</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>1945</td>
<td>Machine language</td>
<td>Translation free</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>High speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Machine dependent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Complex language</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Error prone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tedious</td>
</tr>
<tr>
<td>Second</td>
<td>Mid 1950s</td>
<td>Assembly Language</td>
<td>Easy to use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Faster</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Less error prone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Less efficient</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Harder to learn</td>
</tr>
<tr>
<td>Third</td>
<td>Mid 1950s to early 1970s</td>
<td>FORTRAN, BASIC, COBOL, Pascal, C,</td>
<td>Readability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Machine independent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Less technical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Easier to maintain and document</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Less efficient</td>
</tr>
<tr>
<td>Fourth</td>
<td>1970s onwards</td>
<td>SQL, Oracle Database, Windows Forms, Cold Fusion</td>
<td>Very high-level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Non-procedural</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applicable to specific types of programs</td>
</tr>
<tr>
<td>Fifth</td>
<td>Early 1980s</td>
<td>Natural Languages / AI</td>
<td>Difficult to use effectively</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Current and future development</td>
</tr>
</tbody>
</table>

Table 1.1: Five generations of programming languages [6]

Programming specifically for scientific computation has progressed through the first three generations to a more high-level of abstraction called agents. According to Wooldridge [12], agent systems have evolved from the following five trends that marked the history of computing:

- Ubiquity:
Continual reduction in cost of computing has made it possible to embed computational processing capability into devices and places which were uneconomical and in some cases, even unimaginable. This trend continues, making computational processing capability ubiquitous.

- Interconnection

In addition to the processing capability being embedded into almost every device and place, processors are able to communicate, exchange information and messages with one another to form an interconnected network of large distributed systems. With ever growing internet and hardware capabilities, distributed and concurrent systems have become the norm.

- Intelligence

The complex tasks that can be automated and delegated to computer programs grow exponentially every day. We are progressively gaining a better understanding of the methods of modeling computer programs to deal with more complicated tasks that used to not be at all possible just a few years ago.

- Delegation

The capability to hand over more and more of the computational needs of users to computer systems on their behalf without user intervention is continually increasing.

- Human orientation
The trend to move away from usual machine oriented views of programming towards a more high-level of abstraction that closely relates to the way humans perceive the world, is also on the rise.

From the earliest days of computing to the current trend, programmers have moved from raw machine code to assembler languages, to procedural abstractions, to abstract data types and to the recent state of the art – objects. Each of these developments has allowed programmers to conceptualize and implement programs in terms of always higher-level human oriented abstractions.

![Diagram of software abstractions]

Figure 1. 2: Agent – A human oriented higher-level of software abstraction [13]

Recent trends of delegation and intelligence have given computer systems the ability to act independently to achieve the users’ objectives. Communication and distribution have given the computer systems the capability to cooperate and reach
agreements. All of these trends together have led to the emergence of a new human oriented higher-level of abstraction called ‘Agents’ or ‘Multiagent Systems’ [12].

1.2.2 AGENT DEFINITION

There are many definitions available in the literature for the term “agents”. As D'Inverno [14] points out, it is standard for many researchers to provide their own definition for the term “agent”. In a relatively early collection of papers, several different views emerge. Some of the selected definitions of agents from the literature are as follows:

- Russell [15]: “An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.”

- IBM [16]: “Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in doing so, employ some knowledge or representation of the user's goals or desires.”

- Wooldridge [12]: “An agent is a computer system that is capable of independent action on behalf of its user or owner.”

- Maes [17]: “Autonomous Agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.”
In simple, an agent can be defined as a multi-threaded software entity having its own decision making capability and that takes actions to achieve a set of goals. This piece of code should have the following capabilities as shown in Figure 1.3 [15]:

- Sense its environment
- Decide what action to take
- Execute the action in the environment

![Figure 1.3: Agent – A software entity [15]](image)

### 1.2.3 Agent Environment

Agents perform their actions on the environment, which in turn provides the percepts back to the agent. Design of agent programs are greatly affected by the environments in
which they act. Russell [15] identified principal distinctions between several flavors of environments and suggested the following classification of environment properties.

- **Accessible vs. Inaccessible**: An environment is said to be accessible if an agent can access the complete state information of the environment. An accessible environment is most preferred as the agent need not maintain any internal state to keep track of the changing environment.

- **Deterministic vs. Nondeterministic**: An environment is said to be deterministic if any action has a single guaranteed effect with no uncertainty about the next state of the environment. A nondeterministic environment poses greater challenge in the design of an agent system.

- **Episodic vs. Non-episodic**: In an episodic environment, each agent’s perception and corresponding action together form an episode. Each episode has no link between actions of the agent in different scenarios. An episodic environment is much simpler as the agent need not reason about interactions between current and future episodes.

- **Static vs. Dynamic**: an environment is static if it does not change when an agent is thinking and changes only by the agent’s actions. If the environment changes with passage of time and due to actions or reasons beyond an agent’s control, then it is said to be dynamic. Most of the real-world environments are dynamic in nature and very hard to handle.

- **Discrete vs. Continuous**: An environment is discrete if there are clear, fixed, finite precepts and actions possible in it. Most of the real-world scenarios are continuous.

### 1.2.4 Relevant Work
Micheal [18] argues that the evolution of the term 'agent' metaphor has led to many different uses, which is both a strength and a weakness. The fact that the term has been applied in many different ways, in many circumstances for different purposes is its strength and the weakness being the term agent being used so commonly that there is no general accepted notion of what it is that makes an agent. Micheal [18] addresses this issue by applying formal methods to define a framework for agent systems. This framework considers objects as collections of attributes with a set of action capabilities that can be performed in an environment and that consequently change the state of that environment. Agents are maybe better defined as objects but with some set of goals. Micheal [18] further refines the description of agents and refers to agents with self-motivations or own agendas as autonomous agents.

Taveter [19] introduced layered software architecture, where the software is considered as consisting of three layers: agent at the top tier, objects at the middle tier and a binary layer forms the bottom tier. A similar approach has been followed by Margus [20] in software design, with agents considered as the top-level abstraction units while the agents are implemented using object-oriented programming. Margus [20] do not accept any direct object to object communication in the agent level. The first two levels are mainly software abstractions meant for human understanding and believed to make software development easier, faster and more reliable. Margus [20] argues that the addition of a new layer of agents on top of objects adds value compared to object-oriented approach. This added value is perceived:
• When distributed systems made of autonomous software units are referred to as agents.
• Where common sense is adequate to model software system without going into any technical details.

Soneji [21] applied agent-based optimization approach to military planning for solving allocation problems. This research considers targets and weapons as individual agents in an agent-based environment. Stable Marriage Algorithm (SMA) and Ant-Colony Optimization (ACO) algorithm are used as two agent-based optimization implementation approaches, for solving the weapons to targets assignment problem.

Kicinger [22] proposed a heuristic Method for solving 3D airspace partitioning called GAAB (genetic algorithm/agent-based model). It has two major components namely, a GA for determining the location of agents in the 3D cell-based representation of airspace and an agent-based model for determining cell clustering. First, the GA initializes the population of candidate solutions for the 3D airspace partitioning problem. After initializing a set of agents, the GA computes each candidate's fitness. Each agent represents a sector that is initially composed of a single cell. The agent-based model in turn, determines a clustering of 3D cells. Agents located at their initial locations use their behavioral rules to determine neighboring cells which should be added to their collection of sector cells. After completing the cell assignment process, the quality of cell clustering is evaluated and returned as the fitness value of the solution to the GA.

This evaluation process is repeated for all candidate solutions in the population of the GA. The GA uses its selection mechanism and performs crossover and mutation to
produce the next generation of candidate solutions. The offspring population is again evaluated and the entire process is repeated for a predetermined maximum number of generations. This model was implemented in Java and using several open source toolkits for the GA implementation. Multi-agent simulator of neighborhoods (MASON) provided the framework for agent-based modeling.

Polakow [23] argues that agent-based approaches results in a capability of distributing the computations not only in the physical sense, but also in a logical way. Agent technology in simulation and modeling enables the possibility of an extensive model reconfiguration. Also, it results in a layered structure of the system. Shirantha [24] discusses some of the advantages including:

- Distributes computational resources and capabilities across network of interconnected agents.
- Allows interconnection and interoperation of multiple existing legacy systems into an agent community using wrappers.
- Enhances overall system performance, specifically computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility and reuse.

This chapter introduced packing problem and concept of agents, showed several approaches proposed and applied effectively to solve a variety of optimization problems. In the following chapter, the under hood packing problem is explained in detail, followed by an analysis of how agents and packing problems could be combined into a single system is presented.
CHAPTER 2

PROBLEM FORMULATION

The under hood vehicle configuration design problem with two constraints and three objectives formulated as a function of design variable \( x \) is expressed as:

\[
\begin{align*}
\text{Maximize: } & \{ f_1(x), f_2(x) \} \\
\text{Minimize: } & \{ f_3(x) \} \\
\text{Subject to: } & h_1(x) = 0, g_1(x) \geq \alpha
\end{align*}
\]

Where, \( x \) is the vector of design variables representing the absolute locations of each component, \( f_1(x) \) is the maintainability, \( f_2(x) \) is the survivability, \( f_3(x) \) is the center of gravity height, \( h_1(x) \) is the overlap between components and between components and the enclosure, \( g_1(x) \) is the ground clearance angle of the vehicle, and \( \alpha \) is the minimum ground clearance angle. In the following sections, each of these constraints and objective functions are explained in detail.

2.1 Constraints

2.1.1 Collision Detection

In the packing problem, every combination of two objects must be checked for collision. For a problem with \( n \) objects to be placed, \( \frac{n(n-1)}{2} \) checks have to be performed and the operation is of the order \( O(n^2) \). When the number of components becomes
extremely large, the collision detection process becomes computationally prohibitively expensive. Therefore, it is necessary to have a fast and efficient collision detection algorithm [2]. It is a common practice to start with a bounding box collision detection algorithm for objects that will probably not intersect, eliminating in such cases the need for thorough and time consuming collision detection.

**Bounding box collision detection**

The bounding box collision test is a simple and fast way to perform the collision detection. The bounding box is an imaginary smallest size axis aligned box that encapsulates a given geometry as shown in Figure 2.1. It is represented by two diagonally opposite corner points of the box in space. The bounding box collision detection helps in finding whether a given point is contained by the bounding box.

![Bounding box collision detection](image)

**Figure 2.1: Object and its enclosing bounding box**

Steps involved in bounding box collision detection:
1. The first step in the process consists of constructing a new bounding box object for a given 3D geometry.

2. The second step is the evaluation of whether a point from another 3D geometry is within the bounding box.

The pair of objects shown below in Figure 2.2 is considered to be colliding if any one of the following three conditions is satisfied.

\[
|C_1x - C_2x| < R_1x + R_2x \\
|C_1y - C_2y| < R_1y + R_2y \\
|C_1z - C_2z| < R_1z + R_2z
\]

Figure 2.2: Bounding box evaluation

Where,

$C_1x, C_1y, C_1z$ are the coordinates of the center of the bounding box of object 1
R1x, R1y, R1z are the distances from the center to the bounding box along three axes for object 1

C2x, C2y, C2z are the coordinates of the center of the bounding box of object 2

R2x, R2y, R2z are the distances from the center to the bounding box along three axes for object 2

The bounding box test between the engine and battery for the configuration shown below in Figure 2.3 detects a collision. However, in reality they are not colliding.

![Figure 2.3: Case where bounding box collision detection test fails](image)

Bounding box collision detection has its limitations. Even though it is extremely fast and simple to use, it is not very accurate for complicated geometries which is the case of vehicle component geometries. Hence, a more robust and efficient collision detection method is essential.

To overcome this limitation, the CAD models of components are voxelized into a large number of small volume elements (cubes, cuboid) that best approximate the
component. In this method, geometric data is transformed into voxel data, which helps in performing fast and efficient collision detections. This voxel-based collision detection method is implemented based on the algorithm developed by Tiwari [5]. This algorithm breaks down the problem of components overlap detection to box-triangle overlap detection.

**Steps in Voxel-based Collision Detection**

1. First the bounding box which completely encloses the given object is computed as shown in Figure 2.4.

   ![Figure 2.4: Radiator and coolant reservoir tank enclosed in their respective bounding boxes](image)

2. **Voxelization:** This process involves the conversion of the geometric representation of objects to voxel representation. 3D geometric objects are
fragmented into a large number of small volume elements (cubes, cuboid) that best approximate the continuous object. This is achieved by constructing a three dimensional matrix (database) which entirely confines the bounding box of the object (Voxelization). The speed and accuracy of the collision detection is controlled by the resolution of the voxels and hence their number. For fast, simple and efficient detection, the resolution of the voxels in the X, Y and Z directions are kept constant. However, the implementation still supports varying voxel resolution in X, Y and Z directions. Initially every cell is marked as empty (‘0’) as shown in Figure 2.5.

Figure 2.5: Voxel matrix for radiator and coolant reservoir tank with all cells marked as empty

3. **Facet-box overlap detection**: For every voxel of the bounding matrix, facet-box (triangulated object from Stereo lithography file) overlap detection is performed. If the facet intersects the cell, the cell is marked as non-empty (‘1’) as shown in
Figure 2.6. The triangle-box intersection algorithm is implemented as proposed by T. A. Moller. [4].

Figure 2.6: Voxel matrices after facet-box overlap detection

4. *Voxel inversion:* Voxel inversion is performed to extract the inner volume of the object. To perform this, all the voxels that are outside and on the surface of the object are marked as empty ('0'). Thus, all the voxels which remain marked non-empty constitute the inside of the object as shown in Figure 2.7. The volume of the object obtained from this process is in most of the cases a subset of the actual inner volume of the object. This approximation makes the packing process conservative [2]. The voxel matrix generated for under hood enclosure is referred to as ‘global voxel matrix’.
Figure 2.7: Voxel inversion performed on the voxel matrices

5. *Overlap detection:* To detect an overlap between two objects, the physical coordinates of the bottom-left-back corner of the global voxel matrix is determined. Similarly, the coordinates of all the voxels in the component voxel matrix are determined. Knowing the physical location of all voxels in a component and the global voxel matrix (relative voxel indices i.e., component voxel indices compared to global voxel matrix indices), the global matrix is parsed to check if any voxel in it is occupied by more than one component. If a voxel is occupied by more than one component as shown in Figure 2.8, then those components overlap with each other.
The voxel-based collision detection technique offers a fast and efficient overlap evaluation. The accuracy of the overlap evaluation achieved largely depends on the size of the voxels. The smaller the voxel size is, the more accurate the results can be obtained. This work, which aims at generating good design configuration quickly, considers voxels of size $125 \text{ mm}^3$ ($5\text{mm} \times 5\text{mm} \times 5\text{mm}$ cube) and produces fairly accurate results.

### 2.1.2 GROUND CLEARANCE

Ground clearance has a significant influence on the vehicle dynamics and is defined as the distance between the lowest component of a vehicle and the ground. Higher ground clearance results in a rise in vertical location of the center of gravity, thus resulting in more chance of vehicle rollover and therefore hampers the vehicle stability.

If a component or equipment is lower than the chassis, then the lowest point on that component is used to compute the ground clearance. The ground clearance model defined by Yi [25] is used in this work. According to ISO 612: 1978 [26], ground clearance is
defined by three clearance angles namely, front clearance angle (approach angle, $\alpha_1$), ramp angle ($\alpha_2$) and rear clearance angle (departure angle, $\alpha_3$) as shown in Figure 2.9.

Figure 2.9: Three clearance angles used to compute ground clearance

Ground clearance is defined as the maximum angle of inclination that a vehicle can ascent without the ground scraping against the chassis or any component in the underbody. This process of ascent involves three stages as shown in Figure 2.10:

1. Front tires approach the slope
2. Front tires are on the slope, while the rear tires are on the ground and approach the slope
3. Front tires complete the ascent, while the rear tires are on the slope.
For a vehicle to ascend an inclination without ground scraping, all three clearance angles ($\alpha_1$, $\alpha_2$, and $\alpha_3$) should be greater than the angle of inclination ($\alpha$). Thus, the minimum of these three clearance angles is taken as the ground clearance angle of the vehicle configuration.
2.2 Objectives

2.2.1 Center of Gravity

The height of the vehicle's center of gravity from the ground plays a vital role in vehicle performance, passenger comfort and safety. Lowering the center of gravity will improve vehicle handling and reduce the chance of roll-over [27]. In the under hood, the COG location is determined by the placement of various components and equipment such as body engine, radiator, coolant reservoir, battery, connecting pipes and so on. Technically, the location of the center of gravity in all three dimensions should be considered for the purpose of improving vehicle performance. However, the vertical location of the center of gravity is more prominent when vehicle stability is considered. This problem considers reduction of under hood center of gravity only. As, reduction of center of gravity height in under hood greatly reduces the overall vehicle center of gravity height.

The location of the center of gravity with respect to the ground needs to be maintained as low as possible to prevent the vehicle from over-turning when making turns and to improve the stability of the vehicle. Therefore, the vertical location of the center of gravity needs to be minimized however, the vehicle configuration must be subjected to the minimum ground clearance angle. The overall vertical center of gravity for a system of components is calculated as per the equation

\[ COG = \frac{\sum_{i=1}^{n} (w_i \times COG_i)}{\sum_{i=1}^{n} w_i} \]
Where, $w_i$ is the weight of the $i^{th}$ component, and $COG_i$ is the vertical location of the center of gravity of the $i^{th}$ component and $n$ is the total number of components in the configuration.

Figure 2.11 shows an example vertical center of gravity calculation for a configuration of 5 components. Table 2.1 lists the location of the center of gravity for individual components and their weights.

![Component 1](image1.png) ![Component 2](image2.png) ![Component 3](image3.png) ![Component 4](image4.png) ![Component 5](image5.png)

**Table 2.1**: List of components with weights and vertical location of center of gravity

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight (lb)</th>
<th>COG-Y (units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>3.8</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>2.25</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Figure 2.11: Location of center of gravity from ground

Overall Center of gravity location from ground, $COG = 2.79$ units
2.2.2 VEHICLE MAINTAINABILITY

Maintainability represents the ease with which components can be removed from the under hood. It is a measure of the number of components that have to be removed before the given component can be removed along a selected direction. The simple vehicle maintainability metric proposed by Yi [21] is used. The higher the maintainability metric is, the higher the theoretical vehicle maintainability measure will be, therefore, maintainability needs to be maximized.

The overall vehicle maintainability is defined in terms of the vehicle components accessibility. The vehicle maintainability (M) is defined as

\[ M = A_{max} - A \]

Where,

\( A_{max} \) is the maximum possible accessibility for a given vehicle configuration. It is calculated based on the total number of components and weights assigned to each component. \( A \) is the actual accessibility of the vehicle configuration. This is defined as follows

\[ A = \sum_{i=1}^{n} (w_i a_i) \]

Where,

\( w_i \) is the weight assigned to \( i^{th} \) component which accounts for various maintenance criticalities, since different components may need more accessibility than others for
maintenance purposes. Table 2.2, shows a list of weights assigned to arbitrary components. $a_i$ is the computed accessibility of the $i^{th}$ component.

Figure 2.12 and Figure 2.13 shows an example maintainability calculation for a set of 6 components with the assigned maintainability weights listed in Table 2.2. Case 1 considers ease of removal from an arbitrary “positive” direction and Case 2 from the opposite direction.

<table>
<thead>
<tr>
<th>Component</th>
<th>Maintainability Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2.2: Components and their corresponding maintainability weights

*Case 1: Maintainability along positive X-direction*
Figure 2.12: Configuration with components removed along positive X-direction

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
<th>Components to remove</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Maximum Accessibility, $A_{max} = 31$

Accessibility, $A = \sum_{i=1}^{6}(w_i a_i) = 14$

Maintainability, $M = 31 - 14 = 17$

Case 2: Maintainability along negative X-direction
Figure 2.13: Configuration with components removed along negative X-direction

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
<th>Components to remove</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Maximum Accessibility, $A_{max} = 31$

Accessibility, $A = \sum_{l=1}^{6} (w_l a_l) = 25$

Maintainability, $M = 31 - 25 = 6$

2.2.3 VEHICLE SURVIVABILITY

Survivability is a metric which defines the ability of a vehicle to survive attacks from missiles and bullets. For each component, survivability is the degree of protection provided by overlap with components in the line of fire in the under hood. The higher the survivability index means a better chance for the vehicle to survive attacks, hence the survivability of a vehicle needs to be maximized just like maintainability.

The measure of survivability is similar to the maintainability index as defined by Yi [25] only that survivability depends on the area of overlap with other components instead.
of number of components covering it. With each component having distinct survivability requirements, weights are assigned to components to account for these requirements. The survivability of the vehicle is defined as

\[ S = 0.4S_1 + 0.3S_2 + 0.3S_3 \]

Where \( S_1, S_2 \) and \( S_3 \) are the survivability measures of the vehicle under an attack from the sides, from the rear and from the bottom respectively. The survivability of the vehicle coming from a particular direction is given by

\[ S_j = \sum_{i=1}^{n} (w_i s_i), \quad j = 1, 2, 3 \]

Where \( n \) is the number of components in the vehicle configuration, \( s_i \) is the survivability of the \( i^{th} \) component and \( w_i \) is the survivability weight of the corresponding component. Survivability of the \( i^{th} \) component is determined as below

\[ s_i = O_i / F_i \]

Where \( O_i \) is the total overlap of the \( i^{th} \) component and \( F_i \) is the surface area of the \( i^{th} \) component.

An example survivability calculation for component 1 considering an attack from a direction perpendicular to the plane of paper (from the side for vehicle) is shown in Figure 2.14. In the figure, some part of component 1 is covered by both components 2 and 3.
Steps:

1. Compute the overlap of component 1. Since, region $S_3$ is covered by both components 2 and 3, it is multiplied by a coefficient of 1.5 instead of 2 to favor the solutions that cover more area of component 1 than solutions that cover only a small area but cover the same area many times [25].

$$Overlap_1 = S_1 + S_2 + 1.5S_3$$

$S_1 = 1.95$ sq. units, $S_2 = 1.44$ sq. units and $S_3 = 3.28$ sq.units.

Overlap $= 6.67$ sq. units

2. Compute area of component 1.

$$Area_1 = 35.75$$ sq. units

3. Compute survivability of component 1

$$Survivability_1 = \frac{Overlap_1}{Area_1} = 0.19$$
Figure 2.14: Survivability calculation for three components

Figure 2.15 shows the under hood vehicle configuration design problem formulation for three objectives and subject to two constraints. Objectives are conflicting in nature and a single solution with maximum and minimum objective values cannot be obtained. This work helps the designer in decision making by showing a set of configurations which are non-dominated.

![Diagram of vehicle configuration design process]

Figure 2.15: Under hood vehicle configuration design problem formulation
Having discussed packing problem in detail, the following chapter will address the challenges of restructuring packing problem to incorporate agent-based approach.
3.1 Assessment of Suitable Notion of Agency

The problem of defining ‘what is an agent’ and ‘what makes a program an agent’ has been addressed by many researchers. There is no consensus however on what the proper or universally accepted definition is. Wooldridge [28] considers this question and distinguishes two general usages of the term. The first one being weak and relatively widely accepted and the second one is stronger and possibly more arguable.

Weak notion of agency: This notion is accepted by many researchers and offers a simple way of conceptualizing agents with the following properties:

- **Autonomy**: agents possess the ability to make decisions on their own without intervention from humans or others. This gives the agents control over their possible actions and internal state.

- **Social ability**: agents possess the ability to interact with other agents in a multiagent environment. This interaction may not be merely restricted to simple exchange of bits of information but may extend to cooperation and negotiation between the agents when they have multiple conflicting objectives.
• **Reactivity**: agents maintain an ongoing interaction with their environment and have the ability to respond to changes in their environment by forming a new set of plans to achieve its objectives.

• **Pro-activeness**: Agents not only respond to changes that may occur in their environment, but their response is goal directed. It’s an extension of reactivity in the sense that agents take initiatives and decide on the best way to work to achieve their objectives.

Strong notion of agency: This notion has a stronger and more specific meaning than the weak notion. This notion is generally accepted by researchers working in AI. According to this notion, an agent is a computer system that, in addition to having all the properties of the weak notion, is abstracted and implemented using concepts that are more human oriented. They possess human-like attributes such as knowledge, belief, intention, obligation and emotional states. A more sophisticated agency model has been developed based on this notion by Thorne [29] called BDI (Belief, Desire, and Intention).

According to Jennings [30], the agent technology presents a novel and exciting way of conceptualizing and implementing software. However, it is important to understand some of its limitations:

1. Moving towards strong the notion of agency creates unpredictability in an agent-based system. The source of this emergent behavior is accredited to the sophistication and flexibility given to individual components and their interactions running in parallel.
2. In case of problems with multiple objectives, there can be a significant level of unpredictability about which objective(s) the agent will pursue. Also, under which circumstances, which actions will be employed to achieve the chosen objectives. This further adds to the unpredictability in the agent-system at runtime.

3. Given a circumstance, agents have to first decide among themselves which of their objectives require interactions with which other agents. Hence the number, pattern and timing of interactions cannot be predicted and designed in advance.

4. The dynamics of multi-agent systems are complex and can be chaotic. Hence, agent interaction should be allowed to take place in a controlled environment.

5. Too large a prominence of the agent-specific or intelligence aspects like natural language processing, planning and theorem proving will overload the agent framework. So a more successful strategy to build agents is with minimum AI techniques.

This work considers only the weak notion of agency as the basis for building agent programs as it is simple, very generic, most accepted and applied to wide variety of optimization problems.

### 3.2 AGENT-BASED ARCHITECTURE

Wooldridge [28] argues that they are at least three distinctions between conventional object oriented model and agents. They are:
• *Embed autonomy*: agents have the feature of autonomy being embedded in them. Thus they make their decision on what action to perform for a perceived scenario, unlike objects where one object decides what action to perform for another object.

• *Capable of being flexible*: agents exhibit reactive, pro-active and social behavior. Given any scenario, the agent makes a rational decision of the best possible action that can help in achieving its delegated task and responds by taking a selected action by itself. If not possible to do so on its own, it requests other agents to perform tasks on its behalf.

• *Inherently multithreaded*: a multiagent system is inherently multithreaded, with each agent having its own thread of control.

![Diagram of building agents with object-oriented paradigm](image)

**Figure 3.1: Building agents with object-oriented paradigm**

Objects are passive, which are mostly event driven, or some other object calls a method in them. Unlike objects, agents constantly maintain an ongoing interaction with
their environment. An agent perceives any changes in its environment and responds to it proactively. Although an agent program can be modeled using objects, some of the above mentioned higher-level abstractions are not captured in the object oriented programming model. However, object oriented programming languages like Java have built-in constructs to build multi-threaded components defining weak notion of agency on top of the object model layer.

From the software engineering perspective, functional decomposition techniques have a major influence on designing the software architecture. When applied, it helps in breaking under hood packing software into its lowest functional, non-redundant components which can be easily maintained, modified and reused. These components which have associated intended behavior, tasks, and shared resource usage, are grouped together as agents. This helps in problem decomposition and agent task allocation [31]. By this process, complex interactions among the under hood components are simplified.

This newly developed architecture breaks down the problem into four main classes of agents, namely: configuration generator, objective evaluator, configuration evaluator and component agents. This agent architecture can easily identify and model the responsibilities of each agent and the interactions between them. Figure 3.2 shows the detailed agent architecture, defining each class of agents and their interfaces.
Each agent being derived from the same parent agent class is composed of certain identical properties and methods. However, they differ from each other in their implementation of these common methods and other task specific methods. Figure 3.3 shows the conceptual modeling of four classes of agents, their attributes, inheritance, aggregation and association.

With each component in the under hood modeled as a component agent, the under hood design configuration problem becomes scalable and flexible. This provides the ability to add new components i.e., new design variables to the optimization problem. These agents have their own distinct set of state properties which are self-monitored and updated. This class of agents maintains an ongoing interaction with its environment and exhibit complete cooperation and task delegation. However, these agents interact only
with agents of other classes. The configuration evaluator computes the objective function values for any component agents’ configuration. The configuration generator uses heuristics to modify the current generation of configurations and monitors the evolution of the feasible set of solutions as the constraints are evaluated.

Figure 3.3: Multi-agent system class diagram
A controlled agent environment is created by using a conventional multi-objective optimization algorithm. AMGA2 (Archive-based Micro Genetic Algorithm) controls the multi-agent behavior to reach convergence in the following way [32]:

- It uses very small population size (micro) of twice the number of objectives; this reduces the number of function evaluations exhausted at every generation and hence speeds up the convergence to the pareto-optimal or non-dominated set.
- It decouples parent populations and the current best solutions, thereby allowing independent fine-tuning and selection of two populations.
- It creates a large external archive of non-dominated solutions, giving adequate information about agents’ search history.
- It uses self-adaptive differential evolution operator for crossover and modified polynomial mutation, this strengthens the adaptability and resilience to premature convergence.
- It uses a set of rules to determine algorithm fine tuning parameters, this guarantees good convergence.
The pseudo code for the AMGA2 algorithm [32] is given below:

“Start

Set objectives, desired number of configurations, maximum function evaluations
Generate initial pool of design configurations
Evaluate initial pool of configurations
Add them to external archive
While number of function evaluations are not exhausted
    Select parent pool of configurations from external archive
    Create mating pool from parent pool and external archive
    Generate new pool of design configuration
    Evaluate new pool of configurations
    Update the external archive using new pool of configurations
End while
Display non-dominated configurations from external archive
End”

In Summary, a layered architecture is built with an agent on top of object model. A non-deterministic evolutionary multi-objective algorithm (AMGA-2) is used to identify non-dominated solutions, speed up the convergence to a non-dominated set and prevent unpredictable situations in the agent system. Agents pass problem specific data structures among themselves and take appropriate actions based on the contents of these data structures. Agents are restricted from having too much autonomy to prevent unpredictability in the agent community. AMGA-2 in this context acts like a watchdog and monitors the cooperation and coordination between agents.
This chapter explains how agent-based approach is incorporated into under hood vehicle packing problem as per the weak notion of agency. The following chapter covers results and the benefits of the agent-based approach to the under hood packing problem.
CHAPTER 4

RESULTS

Having setup the agent-based model, components are placed into the under hood one by one, in a predefined packing sequence [8], as shown in Table 4.1. The problem is executed with an initial pool of 250 configurations. From the first generation onwards, the population size is set to 6 and the problem is run for 125 generations i.e., the total maximum number of function evaluations is set to 1000. AMGA-2 is used to help agents speed up the convergence and find the trade-off between the three objectives.

<table>
<thead>
<tr>
<th>Order</th>
<th>Component</th>
<th>Degrees of freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Engine</td>
<td>x, z</td>
</tr>
<tr>
<td>2</td>
<td>Coolant tank</td>
<td>x, y, z</td>
</tr>
<tr>
<td>3</td>
<td>Battery</td>
<td>x, y, z</td>
</tr>
<tr>
<td>4</td>
<td>Radiator</td>
<td>x, z</td>
</tr>
<tr>
<td>5</td>
<td>Boost</td>
<td>x, y, z</td>
</tr>
<tr>
<td>6</td>
<td>Air filter</td>
<td>x, y, z</td>
</tr>
</tbody>
</table>

Table 4.1: Packing sequence with degrees of freedom assigned to each component
Results show that as the survivability increases, maintainability decreases and vice-versa as shown in Figure 4.1. As the three objectives are conflicting, there cannot be a single solution to the configuration design problem, instead a set of non-dominated solutions are obtained. Figure 4.2 shows a plot of non-dominated solutions for three objectives obtained from AMGA-2.
Table 4.2: Objective function values for four examined configurations

<table>
<thead>
<tr>
<th>Configuration #</th>
<th>COG (mm)</th>
<th>Maintainability</th>
<th>Survivability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>483</td>
<td>19</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>475</td>
<td>17</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>482</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>484</td>
<td>10</td>
<td>52</td>
</tr>
</tbody>
</table>

Figure 4.2 and Figure 4.3 show four configurations selected from the set of non-dominated solutions with the objective function values listed in Table 4.2. Of the four selected configurations, configuration 4 has the highest survivability and at the same time lowest maintainability. Configuration 1 has the highest maintainability at the cost of
lowest survivability. This behavior matches with Yi et al.[8]'s results, that vehicle's high survivability metric impedes its maintainability. Configuration 2 has the lowest height of center of gravity while, configuration 3 has better survivability than configuration 2.

Configuration # 1

Configuration # 2

Configuration # 3

Configuration # 4

Figure 4.3: Four vehicle configurations examined

There is no clear winner between these configurations and making a decision is not an easy task. However, this work presents a list of non-dominated solutions to the designer,
who can make the decision based on his previous experience and knowledge or by giving relative importance to an objective over the other.

![Vehicle Packing Result](image_url)

**Figure 4.4: List of non-dominated solutions**

As described in the earlier sections, the agent-based system has certain capabilities that make it best suited for the packing problem. Some of the benefits achieved from the agent-based approach are:

1. **Modularity**: Agents are modeled as proactive objects and hence they share all the benefits of modularity that have led to the success of object technology. The under hood configuration design problem has been assembled from independently developed software modules, having closely connected functionalities. Each module composed of several java classes, are isolated and bundled into separate Java packages. These packages can be exported as an under hood packing API.
Some of the classes in these packages are public and allow to be used in other packing problems.

<table>
<thead>
<tr>
<th>Package</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>packing.constraint.clearance</td>
<td>Computes approach, ramp, and departure clearance angles. Returns the min of these three angles.</td>
</tr>
<tr>
<td>packing.constraint.overlap.level1</td>
<td>Computes axis aligned bounding box for each component, performs bounding box overlap detection. Returns true if components overlap.</td>
</tr>
<tr>
<td>packing.constraint.overlap.level2</td>
<td>Performs Voxelization, triangle-voxel overlap detection. Returns true if any components overlap with each other or if components overlap with under hood enclosure.</td>
</tr>
<tr>
<td>packing.driver</td>
<td>Set of main driver classes. Runs packing problem and contains interfaces and implementation class for AMGA-2.</td>
</tr>
<tr>
<td>packing.geometry.lib</td>
<td>Basic geometric library for Maintainability and Survivability evaluations.</td>
</tr>
<tr>
<td>packing.objective.cg</td>
<td>Computes under hood center of gravity.</td>
</tr>
<tr>
<td>packing.objective.maintainability</td>
<td>Computes maintainability for a given under hood configuration.</td>
</tr>
<tr>
<td>packing.objective.survivability</td>
<td>Computes survivability for a given under hood configuration.</td>
</tr>
<tr>
<td>packing.result.display</td>
<td>Result display package. Shows List of non-dominated solutions. Libraries to display under hood configuration.</td>
</tr>
<tr>
<td>packing.stl.lib</td>
<td>Basic libraries to parse component geometry (Stereo lithography file). Computes component volume, centroid, bounding box.</td>
</tr>
</tbody>
</table>

Table 4.3: Under hood packing Java package
This makes problem reformulation, code editing and adding new functionalities, fast and efficient without making the process messy and overwhelming. These features have significantly improved the software quality by allowing code reuse and reduced software design complexity.

2. **Reusability**: As the agent-based approach is formulated based on software decomposition, it has significantly improved the code reusability and can serve in many different scenarios and applications, avoiding code reimplementation. Developed packing application system can be reused either by
   a. Exporting and integrating it into other systems
   b. Adding new functionalities or tying up multi-disciplinary legacy system as a sub-system to the developed agent-based model.
This feature significantly accelerates formulation of a new packing problem and hence speeds-up development. With further extensions and code revamp, it can lead to a sophisticated concurrent configuration design system.

3. **Flexibility and Scalability**: The above two characteristics of modularity and reusability combine to make the agent-based approach valuable when the problem is likely to change frequently. The agent-based approach incorporates flexibility into the packing problem right from the design phase. It provides the design room to grow and to cater for requirements changes in the future. New objective functions and constraints can be efficiently added to the current under hood packing problem. The packing code can handle additional components and assign degrees of freedom to vehicle components.
To add an objective function:

1. Create a class defining objective function. There should be a method to set component geometry data and design variable (component under hood location). Another method that performs objective function evaluation.

```java
public class MobIndex {
    private Vector bodylist = null;

    public MobIndex() {
    }

    @SuppressWarnings("rawtypes")

    public void setBodylist(Vector vectorBodyList) {
        // set packing problem data structure here
        this.bodylist = vectorBodyList;
    }

    public double Evaluate() {
        // Code to evaluate objective function value goes here
        return 0;
    }
}
```

2. Add the number of objective functions in the main driver.

```
16, //number of design variables
4, //number of objectives
1, //number of constraints
minVar, //lower bound of design variables
maxVar, //upper bound of design variables
1200, //number of function evaluations
600, //desired size of pareto set
randomSeed, //random seed for simulation
debugMode, //verbosity of output (debug level)
evaluator //pointer to the function object - to evaluate a solution
```
3. Create a reference and instantiate objective function class (Mobility Index, an arbitrary objective function created for demonstrating the modularity, reusability and scalability aspects).

```java
// Packing constraint class
BoundingBoxEvaluator overlapEvaluator;
TriangleVoxelOverlapEvaluator overlapDriver;
Clearance clearance;

// Packing objective function classes
MaintainabilityEvaluator maintain;
SurvivabilityEvaluator survivability;
COGEvaluator cogevaluator;
MobIndex mobIndex;

// Packing Problem - Passenger Car Ford Taurus
Taurus taurus;

// Packing constraint class
overlapEvaluator = new BoundingBoxEvaluator();
overlapDriver = new TriangleVoxelOverlapEvaluator();
clearance = new Clearance();

// Packing objective function classes
maintain = new MaintainabilityEvaluator();
 survivability = new SurvivabilityEvaluator();
cogevaluator = new COGEvaluator();
mobIndex = new MobIndex();

// Packing Problem - Passenger Car Ford Taurus
taurus = new Taurus();
taurus.read();
```

4. Set components data and call Evaluate method to perform objective function evaluation and return the objective function value.
Figure 4.5: Steps in adding new packing objective
To add a component to packing problem:

1. Set component geometry and other problem specific component properties (maintainability weight, survivability weight, weight).

```c++
// Newly add component - "Power Steering Reservoir"
path[6] = "\cadmodel\PSReservoir.stl";

// Set maintainability weight for components
MainWeight[6] = 2;

// Set Survivability weight for the component
SurvWeight[6] = 10;

// Set weight of the component
Weight[6] = 60;
```

2. Set bounds on component under hood location

```c++
// Power Steering Reservoir has x, y, z Degrees of Freedom
// Set bounds on the component under hood location
desVar[6] = 3;
minVar[16] = 680-15;
minVar[17] = 1535-100;
minVar[18] = 345;
maxVar[16] = 680+100;
maxVar[17] = 1535;
maxVar[18] = 345+25;
```

Figure 4.6: Steps in adding new component to packing problem
This process scales up the optimization problem by adding new design variables. Even though current work considers the under hood of a passenger car, it can address much larger vehicles like SUVs and trucks. These features make the packing problem flexible and scalable.

4. **Stability:** Since agents have limited autonomy and a non-deterministic multi-objective optimization algorithm controls the communication, cooperation among the agents, there is no source of disturbance that can lead the agent community to anarchical situation. Also, the algorithm monitors and speeds up convergence, hence the modeled multi-agent system always remains stable.
CHAPTER 5

CONCLUSION

5.1 CONTRIBUTIONS

This work solves the under hood vehicle packing problem with three conflicting objectives namely, minimizing vertical the location of the center of gravity, maximizing maintainability and survivability. No overlap between the components and minimum ground clearance form the two packing constraints. Java built-in constructs have been used to realize the weak notion of agency as an additional layer on top of the object model. Classical software decomposition techniques have been carried over and software modules which have related intended behavior, tasks and shared resource usage are grouped together as agents. This architecture helps simulating the complex under hood vehicle component interactions.

In this work, agents pass problem specific data structures among themselves and take appropriate actions based on the contents of these data structures. However, agents are provided with limited autonomy to prevent unpredictability in the agent-based system. A controlled environment is created by using AMGA-2. This non-deterministic, multi-objective algorithm monitors the cooperation and coordination between agents and prevents premature convergence.

As the three objectives are conflicting, there cannot be a single solution to the configuration design problem. However, this work presents a list of non-dominated
solutions to the designer, who can take a decision based on previous experience and knowledge or by giving relative importance to an objective over the other. This work demonstrates the benefits of implementing an agent-based approach to the under hood packing problem based on the weak notion of agency.

5.2 Future work

The developed agent architecture can be significantly improved by integrating it with a formal agent-based framework like Java Agent Development Environment (JADE). This work shows the efficacy of agent-based approaches to reduce the complexity involved in packing problems. Adding GUI to the packing problem will make the application user friendly. User will be able to pick and add vehicle components, provide component properties like weight, color, CAD Model. The user will be able to select a component from the drop down and add it to the problem.
REFERENCES

[1]. Fowler, R. J., Paterson, M. S., and Tatimoto, S. L., Optimal packing and covering in the plane are NP-complete, 1981.


[5]. Santosh Tiwari, Georges Fadel, Peter Fenyes, A Fast and Efficient Compact Packing Algorithm for SAE and ISO Luggage Packing Problems, 2009

[6]. Pierre M. Grignon and Georges M. Fadel, A GA Based Configuration Design Optimization Method


[9]. Vladimir B. Gantovnik, Santosh Tiwari, Georges M. Fadel, and Yi Miao, Multi-Objective Vehicle Layout Optimization


[13]. Eric Yu, Agent-Oriented Modelling: Software versus the World, 2002

[14]. Mark D'Inverno, Michael Luck, Michael M. Luck. Understanding agent systems, pg 3


[18]. Michael Luck and Mark d’Inverno, A Conceptual Framework for Agent Definition and Development

[19]. Taveter, K. and Tamm, B. A new approach to the modeling, design, and implementation of business information systems.

[20]. Margus OJA, Boris TAMM, and Kuldar TAVETER, Agent-based software design.

[21]. Soneji Hitesh Deepak, Comparison of Agent-Based Optimization Approaches Applied to the Weapons to Targets Assignment Planning Problem, 2002.


[23]. Grzegorz Polakow and Mieczysław Metzger, Agent-based framework for distributed real-time simulation of dynamical systems.

[24]. Shirantha Wijesekera, Xu Huang and Dharmendra Sharma, Multi-agent based approach for quantum key distribution in Wi-Fi networks.


[29]. Camilo Thorne, The BDI Model of Agency and BDI Logics, 2005
[30]. Nicholas R. Jennings and Michael Wooldridge, Agent-Oriented Software Engineering, University of London
