Property Inspection Route Planning: What Properties do I have time to visit?

A Minor Thesis submitted in partial fulfilment of the requirements for the degree of
Masters of Information Technology

Christopher John Bond
School of Computing Technologies
STEM College
Royal Melbourne Institute of Technology
Melbourne, Victoria, Australia

June 10, 2022

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

This work has been carried out since 02 March 2021, under the supervision of Dr Maria Spichkova.

Christopher John Bond
School of Computing Technologies
STEM College
Royal Melbourne Institute of Technology
June 10, 2022

Acknowledgements

I extend my thanks to my supervisor Dr. Maria Spichkova, she provided exceptional assistance and support over an extended period of time, and generated a truly captivating research problem. My heartfelt appreciation to my wife Aprille, who may be more relieved than me that I have finished this Minor Thesis. Your belief and support has kept my engine going throughout my many challenges juggling University and full-time work.

Contents

List of Figures

List of Tables

Summary

Finding a place to rent or buy is a complicated quest, planning tends to be effortful and manual. The *Property Inspection Route Planning (PIRP)* problem domain has many features, which makes it impossible to apply existing algorithms directly. In our work, we elaborate two solutions:

- a modification of an existing ILS (Iterated Local Search) algorithm, where we made necessary adjustments to ILS to make it applicable within *PIRP*.
- a novel algorithm, created from scratch and focusing on *PIRP* specifics.

We found that both these algorithms meet the needs of the *PIRP* problem, but the novel algorithm performs with a better score, runs quicker and costs less to operate.

Abstract

Planning inspections to buy/rent is a challenging process. In this Minor Thesis, we formalised this process as a domain called *Property Inspection Route Planning (PIRP)* and placed it in the context of existing route planning domains as a child of the *Orienteering Problem with Time Windows (OPTW)*, with some important differences.

The *PIRP* domain introduces improved travel times accuracy based on modern mapping applications, and establishes the *area inspection* concept, where an individual tours lifestyle amenities in proximity to the property they are inspecting (e.g. supermarkets or restaurants) to help decide on a property.

To address this new domain, *PIRP*, we elaborated two algorithms:

- *PIRP-ILS (PIRP-Iterated Local Search)*, which is our modification of an algorithm commonly used in the parent *OPTW* domain, the goal of the modification is to adjust the original *ILS* algorithm to cover *PIRP* problems and specifics, and
- *PIRP-C (PIRP-Competitor)* a novel algorithm, which we created focusing on *PIRP* problems and specifics from the beginning of its elaboration.

Both algorithms embed a directions API to accurately predict travel time, allow for results customisation from the user, which mitigates scoring inaccuracies and meets the needs of the *PIRP* domain. Our experiments show both algorithms operate within an acceptable runtime, however the novel algorithm *PIRP-C*, performs with a lower runtime, cost and higher score than the adjusted algorithm.

Chapter 1

Introduction

1.1 Motivation

To find a place to rent or buy in a large city like Melbourne in Australia or Munich in Germany is a complicated quest. Searchers visit many properties, which require careful selection and planning of what locations can be visited, what visit duration should be set aside, and what kind of transport can be used.

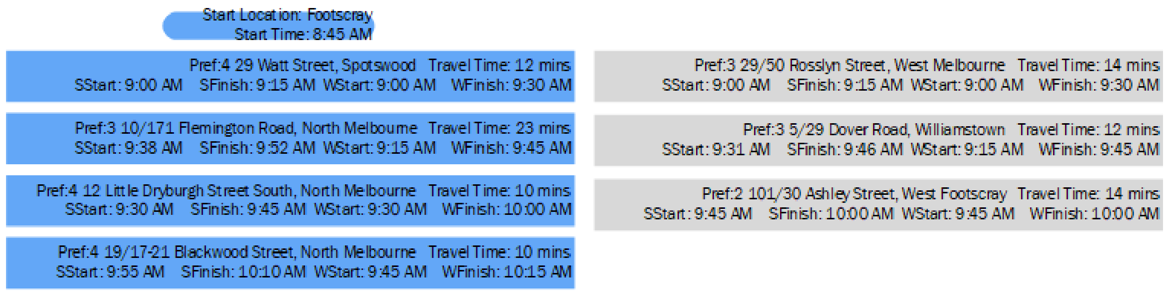
Australian Bureau of of Statistics (ABS) data shows that over 3.5 million Australian residents moved home in 2016, see [5]. Given the significance of the decision, most will not rely on the pictures available on online *Multiple Listing Services (MLS)* [also known as online real estate portals] alone. *MLS* provide assistance with recommendation, but they don't provide any routing and customisation functionality.

Significant decisions, such as real estate inspection planning, require effortful decision making. This can result in fatigue, which leads to poor trade-offs, reduced decision accuracy, lowered satisfaction and increased regret, see [32]. Minimising the complexity of property inspection planning would positively impact the 3.5 million Australian's looking for an abode each year, see [5].

The goal of this project is to relate this problem with existing research domains, analyse differences in property inspection route planning to these related domains, and to provide a solution that will allow for efficient planning of property inspections. Thus, we will assess whether existing journey planning algorithms can be applied to the problem, suggest optimisations, introduce a novel algorithm and a modified algorithm, determine if these algorithms deliver results within an appropriate amount of time, and finally determine which algorithm performs better.

In what follows, we refer to the problem to be solved as the *Property Inspection Route Planning* (PIRP) problem.

We believe that the proposed solution will not only make the search for property to rent/buy more efficient, but also improve the overall user experience. A number of related approaches already demonstrated that this aspect might be very important in the context of route/navigation planning. For example, Chritianto et al. discussed the importance of presentation of choice to users in transportation solutions, further they indicate simple visualisation approaches are far more effective than descriptive approaches, see [11]. As a result, this project will seek to present alternate routes in a simple box-based graphical form, similar to Figure 1.1, to enable simple visualisation and easy user customisation. Unfortunately, other transportation UX research located is rather dated, and therefore will not be leveraged, see [16].



Start Location: Footscray
Start Time: 8:45 AM

Pref:4 29 Watt Street, Spotswood Travel Time: 12 mins
SStart: 9:00 AM SFinish: 9:15 AM WStart: 9:00 AM WFinish: 9:30 AM

Pref:3 10/171 Flemington Road, North Melbourne Travel Time: 23 mins
SStart: 9:38 AM SFinish: 9:52 AM WStart: 9:15 AM WFinish: 9:45 AM

Pref:4 12 Little Dryburgh Street South, North Melbourne Travel Time: 10 mins
SStart: 9:30 AM SFinish: 9:45 AM WStart: 9:30 AM WFinish: 10:00 AM

Pref:4 19/17-21 Blackwood Street, North Melbourne Travel Time: 10 mins
SStart: 9:55 AM SFinish: 10:10 AM WStart: 9:45 AM WFinish: 10:15 AM

Pref:3 29/50 Rosslyn Street, West Melbourne Travel Time: 14 mins
SStart: 9:00 AM SFinish: 9:15 AM WStart: 9:00 AM WFinish: 9:30 AM

Pref:3 5/29 Dover Road, Williamstown Travel Time: 12 mins
SStart: 9:31 AM SFinish: 9:46 AM WStart: 9:15 AM WFinish: 9:45 AM

Pref:2 101/30 Ashley Street, West Footscray Travel Time: 14 mins
SStart: 9:45 AM SFinish: 10:00 AM WStart: 9:45 AM WFinish: 10:00 AM

Figure 1.1: Primary and alternate inspection visualisation

Nielsen indicates that user's stated preferences are not reliably consistent with their true preference. Therefore, it is important to allow for this variability in preference-oriented systems by allowing user-adjustment of solutions, see [37].

Mueller et al. indicate that hedonic scaling (e.g. strongly dislike, dislike, like etc.) is a leading approach in preference measurement, see [35], providing the benefits of:

- understanding closely grouped records
- simplistic entry approach
- flexibility to have no records in a given category
- closely match user sentiment

The main drawback of the above approach is not being able to discern between preference of records within the same category. However, the positives make this approach attractive to the problem area. Another important practical aspect in the context of *PIRP*, is the use of APIs. Despite the wide-scale availability of directions APIs, see [41], leading routing algorithms rarely leverage them, and in many cases resort to euclidean distance, and average-speed approximations in problem solutions, see [45]. An absence of directions API use in routing solutions in the existing literature indicates a risk that directions APIs use may introduce unacceptable delays. Therefore, we consider strategies to limit API calls, namely assessing viability through the euclidean approach above, before assessing the property travel time using a directions API. Incorporating APIs have been considered for the *PIRP* solution to enable more real-world applicability. Similarly to the *PIRP* domain, Zhu and Gonder use map directions APIs to create alternative estimated routes with location pairs and durations, within the domain of cycle detection, see [56]. This approach shows promise for routing in the *PIRP* problem and potential for future work for live plan navigation. A directions API will be tested for applicability for incorporation into the *PIRP* algorithm.

1.2 Real life scenario

Let us illustrate the problem with a scenario: James Smith is looking for properties in Melbourne's inner west, he's selected 20 properties of interest which have inspections on the upcoming Saturday. James knows which properties he prefers over the others, and categorises them as such, he generally takes about 15 minutes inspecting a property and he prefers driving over public transport. James has to manually work out out which properties to select to get to the most inspections he prefers. Figure 1.2 illustrates a manual solution to this problem. The terms WStart and WFinish refer to when the property is open and closed for inspection, whereas SStart and SFinish refer to the 'scheduled' or actual times that James inspects the property. Travel time, preference category (4 being the highest) previous primary property is situated immediately above the respective property.

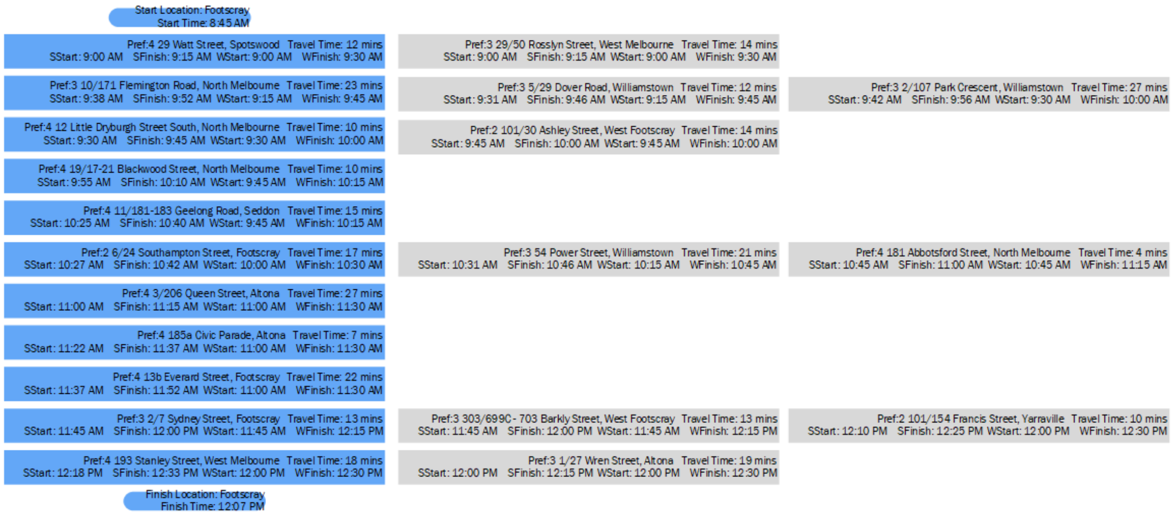

Start Location: Footscray
Start Time: 8:45 AM
Pref:4 29 Watt Street, Spotswood Travel Time: 12 mins SStart: 9:00 AM SFinish: 9:15 AM WStart: 9:00 AM WFinish: 9:30 AM
Pref:3 29/50 Rosslyn Street, West Melbourne Travel Time: 14 mins SStart: 9:00 AM SFinish: 9:15 AM WStart: 9:00 AM WFinish: 9:30 AM
Pref:3 10/171 Flemington Road, North Melbourne Travel Time: 23 mins SStart: 9:38 AM SFinish: 9:52 AM WStart: 9:15 AM WFinish: 9:45 AM
Pref:3 5/29 Dover Road, Williamstown Travel Time: 12 mins SStart: 9:31 AM SFinish: 9:46 AM WStart: 9:15 AM WFinish: 9:45 AM
Pref:3 2/107 Park Crescent, Williamstown Travel Time: 27 mins SStart: 9:42 AM SFinish: 9:56 AM WStart: 9:30 AM WFinish: 10:00 AM
Pref:4 12 Little Dryburgh Street South, North Melbourne Travel Time: 10 mins SStart: 9:30 AM SFinish: 9:45 AM WStart: 9:30 AM WFinish: 10:00 AM
Pref:2 101/30 Ashley Street, West Footscray Travel Time: 14 mins SStart: 9:45 AM SFinish: 10:00 AM WStart: 9:45 AM WFinish: 10:00 AM
Pref:4 19/17-21 Blackwood Street, North Melbourne Travel Time: 10 mins SStart: 9:55 AM SFinish: 10:10 AM WStart: 9:45 AM WFinish: 10:15 AM
Pref:4 11/181-183 Geelong Road, Seddon Travel Time: 15 mins SStart: 10:25 AM SFinish: 10:40 AM WStart: 9:45 AM WFinish: 10:15 AM
Pref:2 6/24 Southampton Street, Footscray Travel Time: 17 mins SStart: 10:27 AM SFinish: 10:42 AM WStart: 10:00 AM WFinish: 10:30 AM
Pref:3 54 Power Street, Williamstown Travel Time: 21 mins SStart: 10:31 AM SFinish: 10:46 AM WStart: 10:15 AM WFinish: 10:45 AM
Pref:4 181 Abbotsford Street, North Melbourne Travel Time: 4 mins SStart: 10:45 AM SFinish: 11:00 AM WStart: 10:45 AM WFinish: 11:15 AM
Pref:4 3/206 Queen Street, Altona Travel Time: 27 mins SStart: 11:00 AM SFinish: 11:15 AM WStart: 11:00 AM WFinish: 11:30 AM
Pref:4 185a Civic Parade, Altona Travel Time: 7 mins SStart: 11:22 AM SFinish: 11:37 AM WStart: 11:00 AM WFinish: 11:30 AM
Pref:4 13b Everard Street, Footscray Travel Time: 22 mins SStart: 11:37 AM SFinish: 11:52 AM WStart: 11:00 AM WFinish: 11:30 AM
Pref:3 2/7 Sydney Street, Footscray Travel Time: 13 mins SStart: 11:45 AM SFinish: 12:00 PM WStart: 11:45 AM WFinish: 12:15 PM
Pref:3 303/699C - 703 Barkly Street, West Footscray Travel Time: 13 mins SStart: 11:45 AM SFinish: 12:00 PM WStart: 11:45 AM WFinish: 12:15 PM
Pref:2 101/154 Francis Street, Yarraville Travel Time: 10 mins SStart: 12:10 PM SFinish: 12:25 PM WStart: 12:00 PM WFinish: 12:30 PM
Pref:4 193 Stanley Street, West Melbourne Travel Time: 18 mins SStart: 12:18 PM SFinish: 12:33 PM WStart: 12:00 PM WFinish: 12:30 PM
Pref:3 1/27 Wren Street, Altona Travel Time: 19 mins SStart: 12:00 PM SFinish: 12:15 PM WStart: 12:00 PM WFinish: 12:30 PM
Finish Location: Footscray
Finish Time: 12:07 PM

Figure 1.2: Manually Determined Inspection Solution

Here we use the following visual notation:

- blue boxes - the properties selected for the visit, as result of the analysis of the provided data;
- grey boxes - these properties were not selected for the visit, but they can be used for an alternative path as "alternative" choices based of the analysis of the provided data;

The aim of our work is to modify an existing algorithm and compare it to a novel algorithm to solve James' problem. Both algorithms will generate a user-customisable route based on preferences, geographic location of properties, the available inspection times, inspection duration and time spent inspecting the local amenities.

1.3 Research Questions

To find a solution to the Property Inspection Route Planning (PIRP) problem, we formulated the following research questions:

RQ1: What parameters are attributes of Property Inspections? Do the PIRP parameters differ from existing route planning problems? If yes, then how exactly do they differ?
To answer RQ1, we elaborated a formal specification of *PIRP*, which we will present and discuss in the next section. Then, in Chapter 2 we will analyse the existing approaches to solve the related planning problems. This will provide us a background for answering the next research question.

RQ2: Can an existing journey planning algorithm be adjusted to meet the parameters defined in the PIRP problem? If yes, then how exactly should it be adjusted? Can a novel algorithm perform better? If an existing journey planning algorithm can't be adjusted, what algorithm will solve this problem?
To answer RQ2, we will analyse the possible adjustments of the algorithm(s) presented in Chapter 2. Based on the results of this analysis, we will either adjust the existing algorithm and create a new algorithm for comparison, or, only create a new algorithm if it is not possible to adjust an existing algorithm. These algorithms are presented in Chapter 3 and the outcomes presented in Chapter 5.

RQ3: Is the performance of the proposed algorithm(s) within the order of magnitude of leading existing routing algorithms? If no, what exactly is the obstacle to achieving the desired performance?
To answer RQ3, we will analyse the leading routing algorithms in Chapter 2 and analyse the algorithm(s) created and/or modified to address RQ2 to confirm whether they fall within the required order of magnitude in the, results described in Chapter 5.

1.4 Outline

The rest of the thesis is organised as follows:

Chapter 2 presents the core concepts of routing problem. We also discuss there the existing solutions that might be promising for adaptation to solve the property inspection problem. The goal of this chapter is to provide an answer to RQ1.

Chapter 3 presents the core results of our work: the proposed algorithm. The goal of this chapter is to provide an answer to RQ3.

Chapter 4 presents the experiments we conducted to evaluate the algorithms. The goal of this chapter is to define the experiment parameters.

Chapter 5 evaluates the core results of our work. The goal of this chapter is to provide an answer to RQ3.

Chapter 6 provides a summary of the thesis.

Chapter 2

Background

The background section focuses on research approach, key concepts and describes the algorithm selected for modification. To identify the related domains, we conducted literature search in IEEE Xplore, Science Direct and Google Scholar databases. IEEE Xplore yielded the lowest number of results with 486, but many of these results were reliable and valid. Science Direct provided 16,498 sources whilst Google Scholar provided 86,460, both these databases provided valuable sources but had many irrelevant topics present. This lead to roughly a similar number of sources being used across the three databases.

The following keywords were applied in the search: *Travelling salesman problem with profits, Tourist Trip Design Problem, Orienteering Problem, Time Windows, Team Orienteering Problem and Directions API.*

We organise this chapter as follows: Section 2.1 provides a general insight into the state of routing problem domain, giving a history of key approaches that have similarities to *PIRP*. This domain was selected because it offers solutions to routing and planning problems like *PIRP*. Following this is Section 2.2 which explores in depth into the single route Orienteering Problem, which is the most promising field for algorithms that may solve the *PIRP* problem. After this, Section 2.3 covers the Team Orienteering Problem which is relevant for multi-day variants, an approach that will be considered for future work. Section 2.4 covers Vehicle Routing Problems, which are a popular domain and the home of several algorithms and benchmark data sets adopted by Orienteering Problems, this domain is not directly relevant, but provide insights into successful approaches in a more heavily researched neighbouring domain. Following this Section 2.5 describes the high level of maturity of recommendation in property inspection websites. Following this, we provide a list of investigated algorithms and shortlist the promising ones. Finally, a description of the algorithm selected and required alterations is covered in Section 2.7.

2.1 Routing Problems

Routing problems are a mature area of research, where real-world applications have led to new domains of research. The core problem refers to maximising value and limiting cost in the context of multiple location of interest (referred to as nodes) with paths that connect them (edges). Each location offers a value, and each path a cost, see [52]. In the *PIRP* context, these locations are properties, and the paths are directions API generated routes between them.

The seminal routing problem is the *Travelling Salesman Problem (TSP)*, which attempts to visit every location while minimising travel cost, see [7]. An evolution of this idea is referred to as the *Travelling Salesman Problem with Profits (TSPP)*, it introduces varying value for each location, and eliminate the requirement to visit every location. *TSPP* retains the dual goals of maximising value, while minimising cost, see [15].

This concept spawned the *Orienteering Problem (OP)*, see [52], which considers the travel budget as a fixed roof, and the *Vehicle Routing Problem (VRP)*, see [4], which introduces parameters of road navigation. These problems have established variants that consider *time windows*, including the *(Team) Orienteering Problem with Time Windows ((T)OPTW)*, see [21], and the *Technician Routing and Scheduling Problem (TRSP)*, see [33], respectively. *Time windows* are start and finish times, within which the visit must be made to qualify as valid. These problems solve the issue of scheduled technician appointments and point of interest opening-hours. They tend towards large windows and short engagement time, resulting in several ordering options for the route. Unfortunately, exact solutions in these fields are stated to be unrealistic for real-world application beyond a few nodes, due to high-order polynomial behaviour, see [17, 33, 22, 45]. Therefore, solutions focus on approximation algorithms that solve in lower-order polynomial time.

2.2 Orienteering Problem

The *Orienteering Problem (OP)*, introduced by Tsiligirides, varies from the *TSP* by converting the travel budget into a fixed constraint, where the *TSP* seeks to minimise travel cost, see [48]. The OP more closely resembles the *PIRP* problem, given inspections are likely to be carried out over the course of a reserved day and should be maximised as opposed to the *TSP* approach, which assumes every property must be visited in the shortest time possible. Conceptually, Gunawan et. al. describe the OP as a combination of the classical *Knapsack* and *Travelling Salesman Problems*, see [21], which suggests that algorithms that perform well for solving both problems should also solve the combined problem, a class of algorithms regularly applied in both domains are evolutionary algorithms, as presented in [44, 51, 1, 52, 13, 45].

The *orienteering problem* has variants with *time windows (OPTW)*, for point of interest open-close times, and *time dependency (TDOP)*, for public transport usage. *Time windows* closely resemble property inspection times in the *PIRP* problem. The orienteering problem represented in Figure 2.1, is an edge-weighted graph where preference is represented by radius of each circle, the selected path is made up of orange property locations (nodes), arrow paths (edges) [travel time length] and green start and end locations. Further, available properties, not selected are represented in blue, based on [17]. Leading algorithms that directly solve this problem are *Discrete Strengthened Particle Swarm Optimisation* [44] and *Greedy Randomized Adaptive Search Procedure and Path Relinking* [9]. While these algorithms efficiently solve

OP, they ignore time windows. This means a fundamental shift in algorithm construction would be required to make them relevant for our shortlist.

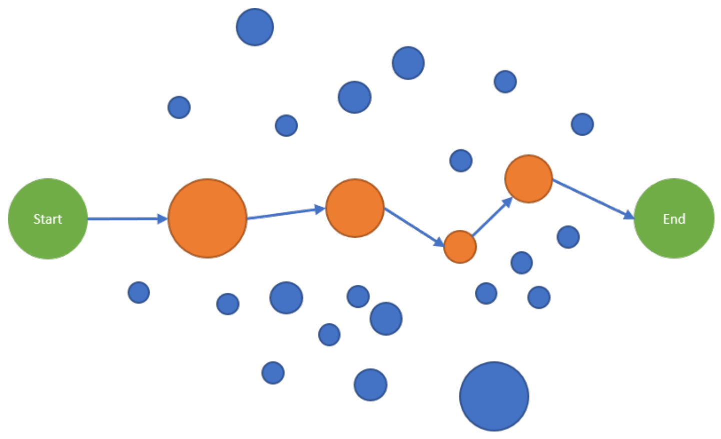


Figure 2.1: Single tour Orienteering Problem

The *tourist trip design* sub-problem of *OP* includes 3 elements: *Recommendation, Route Generation and Customisation Functions*, see [17]. The *customisation functions* focus on adding or removing points of interest. Similarly in our research, we seek to provide customisation, allowing users to substitute selected properties with alternatives, so an algorithm with proven integration with customisation is promising. Algorithmic *route generation* is the focus of this thesis. *Property recommendation* is mature functionality for *MLS*, see [49], so inclusion would not represent a novel development, however integration with recommendation services would make future solutions more relevant to the the *MLS* user base, so will constitute future work.

As introduced earlier the *tourist trip design* sub-problem focuses on the *OPTW* variant, which has relevance to the *PIRP* problem. *Iterated Local Search (ILS)* was presented as a fast approach to solving *OPTW* problems, see [30], while Gunawan et al. [20] presents a more accurate, albeit computationally intensive solution approach, through an adjusted ILS algorithm. Given the *OPTW* is a close match to the *PIRP* problem, *ILS* has been chosen as the algorithm for use as the base-case *PIRP* algorithm defined in Section 3.4.

2.3 Team Orienteering Problem

The *team orienteering problem (TOP)* enables multi-day [17] or multi-agent [22] scheduling by extending the single-path approach of *OP* into multiple paths, this was first introduced by Chao et. al. [10] . A pareto mimic algorithm was introduced by Ke et al. [25] that solves *TOP*, additionally Li et al. [27] put forward a dynamic programming algorithm and Verbeeck et al. [51] submitted an *ant colony system (ACS)* for the *time dependent team orienteering problem (TDTOP)*. These algorithms miss the fundamental *time window* parameter that is required by *PIRP*. Even more relevant are the adapted genetic algorithm submitted by Abbaspour et al. [1] and the *ant colony optimisation (ACO)* algorithm created by Verbeeck et al. [52], which address *time dependence and time windows*. Given the focus on multi-dayagent scheduling problems, *TOP* are not as relevant to the *PIRP* problem as *OP* algorithms, because *PIRP* is a single-agent and single-day focused problem in this minor thesis. *TOP* approaches may be relevant in future work exploring multi-day plans.

2.4 Vehicle Routing Problems

Technician Routing Problem (TRP) Mathlouthi et. al. [33] present an advanced algorithm to solve *TRP*, that addresses a large number of attributes, several that are relevant to *PIRP*, for example time windows. However, many attributes are not relevant, for example skills, overtime, food breaks, tool and inventory use and equipment depots. The authors do not address algorithmic complexity, however the growth rate in wall-time from 20+ nodes (properties), suggests quadratic complexity, which makes this algorithm less competitive.

Time Dependent Vehicle Routing Problem (TDVRP) Gmira et al. [18] present a Tabu Search algorithm that solves the Time Dependent Vehicle Routing Problem, which does not address the time window attributes of the *PIRP* problem.

Vehicle Routing Problem with Time Windows (VRPTW) Shen et al. [45] describe a hybridised algorithm called *ant colony system with brain storm optimisation*, which addresses the *VRPTW. TRP, TDVRP and VRPTW* algorithms aim to minimise overall distance travelled, which makes their aim fundamentally different to both the *PIRP* and *OP* domains, which seek to set an overall time limit and to maximise value. This undermines their applicability to the *PIRP* problem due to fundamental changes in operation required. The complexity of this approach and variance from the *PIRP* domain means that a variant of these algorithms has not been considered for the *PIRP* domain.

2.5 Recommendation Engine

All the algorithms explored above treat property(location)-weighting as a necessary process but out of scope for their algorithms. Given this gap, optimal list curation approaches were investigated. Yuan et. al. leverage image metadata to recommend routes based on time data, see [55], Lim et. al. [29] agreed and further utilised the data to recommend visit durations. It was found that *MLS* provide recommendation services that adequately address the recommendation function, with a high degree of maturity. Further, *MLS*'s encourages users to save preferred properties to *customisable lists*, see [49]. For the purpose of this project we will assume these lists can be categorised with relative ease, and this will be used as an input to the *PIRP* algorithm. However, integration with recommendation functions will be considered in future work.

2.6 Preliminary analysis of available algorithms

Algorithms have been assessed and shortlisted based on their problem similarity and features. The following algorithms were not shortlisted because they failed to address the *PIRP* domain:

- *DStPSO and GRASP* solve OP, which ignores the crucial Time Window element, see [44, 9].
- *Pareto Mimic algorithm, Ant Colony System and Adapted Genetric Algorithm* similarly ignore time windows while solving the potentially useful Time Dependency problem, see [25, 27, 51].
- *Branch and Price Algorithm, Tabu Search and Ant Colony System with Brain Storm Optimisation* all solve variants of VRP who's focus on minimising makespan is fundamentally different to the *PIRP* problem, efforts to change them would be significant, see [33, 18, 45].

The following algorithms were shortlisted for assessment:

- *Iterated Local Search (ILS)* demonstrated fast run-time, and is leveraged as a component of leading strategies [21, 33, 17] Fast run-time may be particularly relevant because of the likely use of a directions API.
- *Hybridized Simulated Annealing - Iterated Local Search (SAILS)* demonstrated high degree of accuracy, with additional computational overhead, which may limit the use of directions APIs for travel time, see [2].
- *Ant Colony Optimisation (ACO)* shown promise with a large number of constraints, in routing and scheduling.[13]
- *Adapted Genetic Algorithm* shows promise for a complete service algorithm which would not leverage a directions API due to its solution to *time dependency.* It provides a solution with moderate computational overhead and accuracy, see [1].
- *Ant Colony Optimisation* similarly shows promise for a complete service algorithm, again with moderate computational overhead and accuracy, see [13].

2.7 Selected Algorithm

The *improved iterated local search (ILS)* algorithm from Gunawan et al. [20] that addresses the *Orienteering Problem with Time Windows (OPTW)* domain, was selected for modified application in the *PIRP* domain.

The following subsections are laid out as follows:

- Subsection 2.7.1 covers the reasons why *ILS* was selected, while partially answering Research Question 1, laid out in Chapter 1.

- Conversely, Subsection 2.7.2 lays out the reasons why alteration to this algorithm is required to address the *PIRP* domain.
- Subsection 2.7.3 lays out a definition of Gunawan et al. [20] ILS algorithm with notations changes that reflect the *PIRP* domain.

2.7.1 Reasons for selection

ILS was selected because it addressed several key requirements:

- *Orienteering Problem (OP)* – Conducts inspections through a selected day, and each property has a varying value to the user, which accords with the *OP*, which this algorithm solves.
- *Time Windows* – Allowing inspection start and finish times, which can be addressed through an algorithm that solves the *OPTW*, like this algorithm.
- *Single route* – Chosen for simplicity and speed for initial implementation.
- *Accurate* – The algorithm performs well, able to improve upon 8 of the best-known solutions as compared with domain benchmark instances, see [20].

2.7.2 Reasons for alteration

We altered the *ILS* algorithm for the *PIRP* domain because it did not address the following:

Practical Transport Times Transport times in existing approaches use naive calculations. Straight-line (Euclidean) distance between coordinates are used to estimate travel times based on a predetermined speed. Whereas users routinely experience free, fast and accurate routing directions from routing services such as Google Maps. Alvarez et al. demonstrate time discrepancies of up to 11% when comparing dynamic congestion routing using the Google directions API as compared with Euclidean distance-time, see [3]. This could have a significant impact on inspections chosen in our scenarios, where real estate inspections are relatively narrow time windows as compared with classic *OPTW*. A narrower inspection window can lead to small travel time differences making certain inspections, no longer viable, this is discussed in detail in Section 3.5.2. Given the maturity and accuracy of routing services, a directions API should be leveraged to estimate travel times between properties, however any use should be offset by changes in the algorithm to avoid significant increases in calculation time over existing algorithms. We see in Luthfie et al. [31] that the Google Maps API delivers responses with a latency between 0.326 - 0.784 seconds for a request, assuming serial requests and with a n^2 time complexity (as discussed in section 3.5.4), with 50 properties we calculate an additional runtime of 13 - 32 minutes. Google Maps has been chosen as the routing service because it claims to provide the most accurate Estimated Times of Arrivals of API mapping platforms, see [14].

Multiple inspection types In the *OPTW* domain all inspections must occur within time windows, however in the *PIRP* domain we introduce an additional class of inspections, those not bound to *time windows*. These are defined as *area inspections* and occur in addition to time window bound inspections. The requirement for this is based on an individual gaining an understanding of the local amenities in the area surrounding a property of interest. In practice, any two or more properties within vicinity of each other would only require one inspection of local area amenities.

Multiple travel modes Typically, euclidean travel times assume car-based travel as universal for all routes. In some locations or life-situations, public transport may be preferable to car travel. Further for short travel distances, taking public transport or moving a parked car a short distance could be unrealistic as compared with walking. For these reasons, options for public transport or car transport are allowed for, and walking is selected for sub-10 minute travel times.

Score importance verified Gunawan et al. explain that for *OPTW* better results are experienced when the score value is squared (u^2), to weight it heavier than the duration, see [20]. In the *PIRP* domain, it is not certain whether the more restrictive the time window will require a similarly increased score weighting. Therefore, three scenarios are explored with the score sensitivity kept at u^2, and adjusted to u and $\sqrt{u}$.

2.7.3 Original Algorithm Definition

What follows are the key elements that make up the *ILS* algorithm, see [20], with minor notation changes that apply to the *PIRP* domain. For example, nodes are referred to as properties. An initial solution is created through a greedy heuristic, then enhanced via Iterated Local Search (ILS), made up of LocalSearch, Perturbation and AcceptanceCriterion, all steps are outlined below:

Algorithms - Greedy Construction

Simplified Summary A simplified view of the key construction steps are shown in Figure 2.2 below. To iteratively select each viable property, each remaining property is considered from the start location and start time. The amount of travel and inspection time is considered in minutes (in the orange boxes), and if viable (i.e. we are able to make it to the location and conduct the inspection in full before the window closes) the locations are included in a shortlist (our illustration uses 3, *ILS* uses 5). The score (in dark blue boxes) is then divided by the time taken to produce a ratio (step 2 in Figure 2.2). Each ratio is summed, then divided by the sum in step 3. Finally, in step 4, a roulette wheel is constructed from the ratios, and a random number generator is used to select the next scheduled location. The black line indicates the starting position and the red line indicates the random number output, in this situation the output is 0.41, which selects $p1$. The algorithm will then proceed, using the finish time and location of $p1$ as the new start location and start time, to consider the remaining properties.

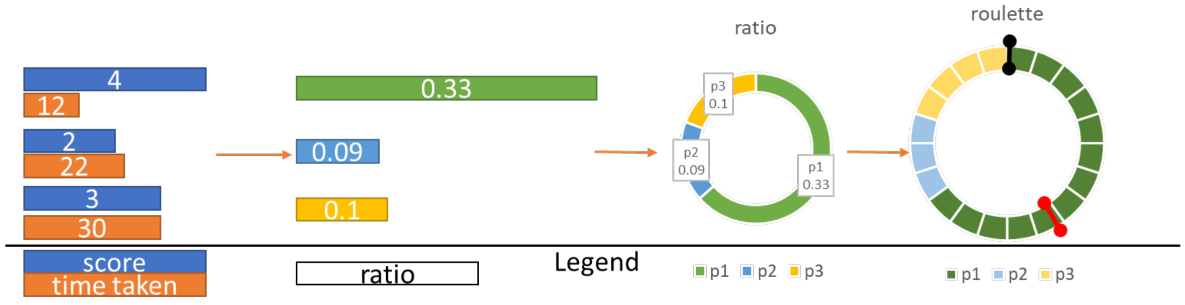


Figure 2.2: Simplified ILS selection Illustration

Detailed Explanation Starting with a blank solution, properties are added iteratively to the solution until no longer viable, as outlined in Algorithm 1. P' are unscheduled properties and $P*$ the scheduled properties, $P' + P* = P$ (all properties) and V are the viable candidate properties to be inserted. $P*$ is initialised by P_{start} and P_{end}, the start and finish locations also known as $HomeLoc_p$. S_0 refers to the current viable solution.The set V is iteratively generated to store viable candidate unscheduled properties to be inserted into S_0. We examine all possibilities of inserting an unscheduled property in serial location l. Insertion of a property is allowable if the scheduled properties after an insertion fit within their *time windows* and the total inspection and travel time does not become greater than T^{max}.

Algorithm 1 Construction

```
 1: $P* \leftarrow P^{start}$ and $P^{end}$                         ▷ scheduled properties
 2: $P' \leftarrow P \setminus P^{start}$ and $P^{end}$             ▷ unscheduled properties
 3: Initialise $S_0 \leftarrow P*$
 4: $V \leftarrow$ UPDATEV($P'$)
 5: while $V \neq 0$ do
 6:     $\langle p*, l*\rangle \leftarrow$ SELECT($V$)
 7:     $S_0 \leftarrow \langle p*, l*\rangle$
 8:     $P' \leftarrow P' \setminus \{\ p*\}$
 9:     $P* \leftarrow P* \cup \{\ p*\}$
10:     $V \leftarrow$ UPDATEV($P'$)
11: end while
12: return $S_0$
```

Each instance within V, which represents a viable insertion of property p in serial location l, is represented as $\langle p, l\rangle$. At insertion, the advantage of insertion $ratio_{p,l}$ is computed utilising equation 2.1. $\Delta_{p,l}$ is the change in total time spent prior to and after property insertion at location l. E.g. the sum of time spent on the route increases from 870 minutes to 900 minutes after property insertion, $\Delta_{p,l}$ is 900 - 870 = 30 minutes. All elements of V have $ratio_{p,l}$ calculated and are sorted in descending order, only v elements are retained. V is generated using Algorithm 2.

$$ratio_{p,l} = \frac{u_n^2}{\Delta_{p,l}} \tag{2.1}$$

If V is not empty, Algorithm 3 is executed to choose $\langle p*, l*\rangle$ to be inserted. Each $\langle p, l\rangle$

Algorithm 2 UPDATEV

```
1: $V \leftarrow 0$
2: for all $p \in P'$ do
3:     for all $l \in L$ do
4:         if insert property $p$ in location $l$ if viable then
5:             calculate $ratio_{p,l}$
6:             $V \leftarrow V \cup \langle p, l \rangle$
7:         end if
8:     end for
9: end for
10: Sort all element of $V$ in descending order based on $ratio_{p,l}$
11: Select the best $v$ number of elements of $V$ and remove the rest
12: return $V$
```

matches with a probability value $prob_{p,l}$ which is computed in Equation 2.2:

$$prob_{p,l} = \frac{ratio_{p,l}}{\sum_{(i,j,k)\in F} ratio_{i,j,k}} \tag{2.2}$$

Choosing $\langle p^*, l^* \rangle$ from V uses Roulette-Wheel selection, see [54]. The probability of choosing a candidate insertion which is proportional to benefit $ratio_{p,l}$. The probability values sum, $SumProb$, start at 0. A random value is created U $[0, 1]$, where 1 is the total $SumProb$. A candidate $\langle p^*, l^* \rangle$ is selected and $SumProb$ is updated. The loop concludes when ($U \leq SumProb$) then the respective $\langle p^*, l^* \rangle$ is chosen. S_0, P' and P^* is updated post insertion. Greedy construction concludes on the exhaustion of viable candidates, on $V = 0$. Gunawan et al. claims that the score for candidate insertion in the $OPTW$ domain is a more important factor than candidate time expenditure, see [22]. For this reason the score value is squared in Equation 2.1, this increases the probability that a higher scored property will be selected in Algorithm 3, however this minimises the effect of inspection and travel time on the algorithm.

Algorithm 3 SELECT V

```
1: $SumRatio \leftarrow 0$
2: for all $\langle p, l \rangle \in V$ do
3:     $SumRatio \leftarrow SumRatio + ratio_{p,l}$
4: end for
5: for all $\langle p, l \rangle \in V$ do
6:     $prob_{p,l} \leftarrow \frac{ratio_{p,l}}{SumRatio}$
7: end for
8: $U \leftarrow rand(0, 1)$
9: $SumProb \leftarrow 0$
10: for all $\langle p, l \rangle \in V$ do
11:     $SumProb \leftarrow SumProb + prob_{p,l}$
12:     if $U \leq SumProb$ then
13:         $\langle p^*, l^* \rangle \leftarrow \langle p, l \rangle$
14:         break
15:     end if
16: end for
17: return $\langle p^*, l^* \rangle$
```

Algorithm 4 ILS (P,R)

```
 1: $S_0 \leftarrow$ CONSTRUCTION$(P, R)$
 2: $S_0 \leftarrow$ LOCALSEARCH$(S_0, P^*, P', R)$
 3: $S^* \leftarrow S_0$
 4: NOIMPR $\leftarrow 0$
 5: while TIMELIMIT has not been reached do
 6:     $S_0 \leftarrow$ PERTURBATION$(S_0, P^*, P')$
 7:     $S_0 \leftarrow$ LOCALSEARCH$(S_0, P^*, P')$
 8:     if $S_0$ better than $S^*$ then
 9:         $S^* \leftarrow S_0$                              ▷ new solution becomes best solution
10:         NOIMPR $\leftarrow 0$                             ▷ NoImpr becomes 0
11:     else
12:         NOIMPR $\leftarrow$ NOIMPR $+1$                   ▷ iterate NoImpr
13:     end if
14:     if (NOIMPR+1) MOD THRESHOLD $= 0$ then                ▷ threshold no. of iterations
15:         $S_0 \leftarrow S^*$
16:     end if
17: end while
18: return $S^*$
```

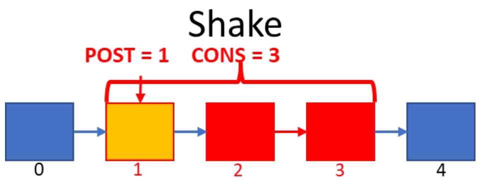
Algorithm - Iterated Local Search

Below, enhancements to the solution S_0 created in the construction phase outlined are detailed. *Iterated Local Search (ILS)* is the overall algorithm, it employs CONSTRUCTION then uses PERTURBATION, LOCALSEARCH and ACCEPTANCECRITERION functions to improve the solution. The *ILS* algorithm is displayed in Algorithm 4.

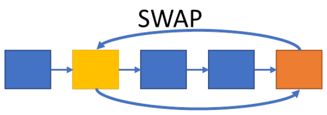
The PERTURBATION step is a single SHAKE operation. SHAKE attempts to escape the local optima, illustrated in Figure 2.3a, leverages two integers. 1 - CONS is the number of consecutive properties to be removed. 2 - POST is the index of the first of these properties. Upon reaching the final scheduled property, the procedure loops back to the start property. CONS and POST are initialised as 1, on each SHAKE, POST is then increased by the value of CONS. Upon a fixed number of iteration CONS is increased by 1. Where CONS grows equal to or greater than $P/3$ (property count), CONS is reset to 1. Note that Gunawan et al. [20] described this last step generally, rather than algorithmically, but offers a detailed modified algorithm in a sibling paper that compares multiple routes, submitted to solve TOPTW, see [22]. This algorithm is drawn from Vansteenwegen et al. [50], which is the source Gunawan et al. [20] base their algorithm variations upon.

In LOCALSEARCH, 4 procedures are executed in series. SWAP and 2-OPT attempt to reduce the overall time taken on the route, without reducing the number of properties seen. These prepare for INSERT and REPLACE, which attempt to increase the overall score. First SWAP, illustrated in Figure 2.3b is used to exchange any combination of two scheduled properties.SWAP is executed if it reduces overall route travel time, and constraints are not contravened. Secondly, 2-OPT, illustrated in Figure 2.3c selects two scheduled nodes and reverses their sequence (given a constraint contravention doesn't occur, and there is an improvement overall route travel time). Note that in the context of *PIRP*, with narrow time windows, these two procedures are less likely to add value, as each inspection has a narrower time window in which to be adjusted. Third, INSERT, illustrated in Figure 2.4a puts one unscheduled property into the route. Initially, V is generated using Algorithm 2 and inserted with Algorithm

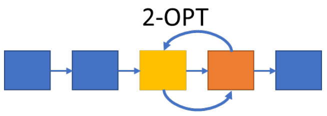
3. This is continued until V = 0. Finally, Replace, illustrated in Figure 2.4b attempts replacement of one scheduled property with one unscheduled property (from a list sorted by the highest score). It is accepted if the solution is viable and the total score is improved. If accepted, the next unscheduled node is considered. If infeasible, the operation is terminated. If better than the *best* solution $S*$, this new solution S_0 is accepted as the *best* solution. The current solution S_0 is submitted as the starting solution for the next iteration of local search. However a limit is introduced by Acceptance criterion, if no improvement is experienced for the solution over a certain number of Threshold runs, the current solution S_0 reverts to the latest *best* solution $S*$. Additionally, no further runs are executed once the algorithm exceeds TimeLimit.



(a) Shake

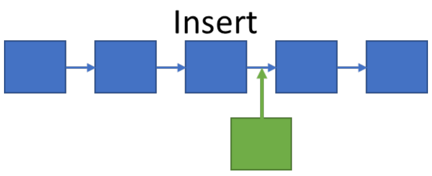


(b) Swap

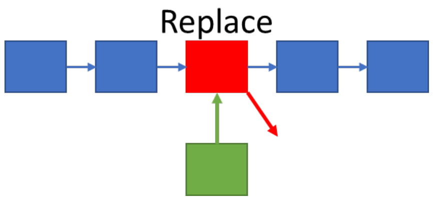


(c) 2-OPT

Figure 2.3: Scheduled Inspection Swapping Operations



(a) Insert



(b) Replace

Figure 2.4: Removing Scheduled Inspection Operations

2.8 Summary

In this chapter we introduced various routing problems, highlighting that the *Orienteering Problem*, *Team Orienteering Problem* and *Vehicle Routing Problem* were relevant to the domain we were introducing, we then detailed these domains in their own sections. Next, we indicated the *Orienteering Problem* includes the *Tourist Trip Design Problem*, which is very similar to our proposed domain. Following this, we described the *Team Orienteering Problem* as an *OP* one conducted with multiple people or over multiple days, which is not relevant to the *PIRP* domain. Next, *Vehicle Routing Problems* are explained, with the time windows variant most applicable, but still not readily relevant to the *PIRP* domain. We then highlight that *recommendation engine* are a present feature of *MLS*, where our data is sourced. We then analyse several algorithms before selecting *ILS* for alteration in the *PIRP* domain and describe in detail the selection reasons, alterations required and detailed operation of *ILS*.

Chapter 3

Proposed Algorithms

This chapter formalises the *PIRP* problem, outlines the assumptions used to formulate the solutions, highlight the differences from classic OPTW solutions and introduces two version of the proposed algorithm. First an adjusted algorithm *PIRP-ILS* and novel algorithm *PIRP-C*. Finally, Section 3.6 summarises the chapter. The evaluation of the algorithms will be presented in the next chapter.

Both solutions are based on *ILS*, presented by Gunawan et. al. [20]. As mentioned in Section 2.7.2, Gunawan's *ILS* algorithm does not utilise a directions API (and therefore practical transport times and flexible travel modes are not available), multiple inspection types are not provided for, and the score value is maximised at the expense of the time value.

Both algorithms we propose in this chapter may be promising in solving the *PIRP* problem. *PIRP-ILS* retains the original algorithm approach, adding key functionality of an additional inspection type, multiple modes of transport, a more accurate travel time calculation and displaying of alternative properties. *PIRP-C* contains the same additions, but uses a novel approach to algorithm construction and improvement, reducing the number of comparisons and therefore the potential time impact of the additional functionality. The aim of this chapter is to introduce these algorithms and compare them.

3.1 Formalisation of PIRP

This section defines the *property inspection route planning (PIRP)* problem description. This problem description is made up of core criteria, attributes and constraints. Given *PIRP* is a subset of *OPTW*, the constraints closely follow Gunawan et. al. *OPTW* description, see [20]. Further, the constraints are separated into similarities and differences. The *PIRP* problem has the following core criteria:

- *User preferences*: The value to the user in the suggested solution is near-optimal.
- *Quick runtime*: The runtime is minimal, and should be at most, equivalent to *ILS* runtime.

- *Alternative properties*: Presented to the user for selection alongside suggested property route.
- *Matches constraints*: The solution abides by the constraints set out in Section 3.1.2.

3.1.1 Attributes

In *PIRP*, we require a set of general user parameters for solving the problem, further an array of properties each with a set of specific property parameters is needed. The mix of selected properties will determine the overall score delivered by an algorithm to the *PIRP* problem. The general user parameters for the properties potentially selected for inspection will be denoted by $PSet_p$.

For creating a plan, we will need the following general parameters:

- Dur_p – Inspection duration
- Day_p – Day of inspection
- $Mode_p$ – Travel mode selected
- AD_p – Default duration inspecting the local area (before or after inspection).
- ST_p – Start time for the plan
- FT_p – Finish time for the plan
- $HomeLoc_p$ – Initial location of the person. This location is not displayed in the results set. This is considered both:
  - P^{start}, referring to the start location in the plan.
  - P^{end}, representing the finishing location in the plan. We assume that the individual will want to return to the location that they started from.
  - $PList_p$ – The

Thus, a plan will be specified as a tuple

$$PSet = \langle Dur_p,\ Day_p,\ Mode_p,\ AD_p,\ ST_p,\ FT_p\ HomeLoc_p,\ PList_p \rangle$$

where $PList$ will be a list of selected properties $[P_1,\ ...,\ P_n]$, where for each i, $1 \leq i \leq n$, $P_i \in PSet$.

Consider a set P of properties. Each property inspection $i \in\ P$ has the following attributes:

- GL_i – Geographic location of the property presented by the address, longitude and latitude.
- u_i – Preference category (or score) per property.
- TT_{hi} – Travel time between property h (previous property) and i. Note this is non-negative.

- AD_i – Area duration of the property i.
- $[e_i, l_i]$ – Inspection time window, defined by the agent, logically $e_i < l_i$, where:
  - e_i – Opening time for an inspection at property i.
  - l_i – Closing time for an inspection at property i.

Per default, when an inspection is created it belongs to the set of *unscheduled inspections*, until it's added to the plan $Plan$, i.e., $scheduled(i) = false$ and $i \notin Plan$, where
$I \notin Plan \ \rightarrow \ \neg scheduled(i)$.
This also mean that some parameters, specific for scheduled inspections (such as the times of scheduled start and finish) will be undefined and have value $UNDEFINED$.

Once an inspection has been added to the plan it is considered a *scheduled inspection*, this adds the parameters *Scheduled Start* S_i^{Plan} and *Scheduled Finish* F_i^{Plan}:

- $i \in Plan \ \rightarrow \ scheduled(i)$,
- $S_i^{Plan} < F_i^{Plan}$,
- $S_i^{Plan} \geq S_i$, and $F_i^{Plan} \leq F_i$.

Until an inspection is added to the plan, the values of S_i^{Plan} and F_i^{Plan} are undefined.

Equation 3.1 sets the area duration AD_i to 0 if a property in the area has previously been visited, otherwise it is set to the standard area duration setting for the region that the plan is conducted within. This equation ensures that an additional inspection of the local amenities (area inspection) is conducted if an inspection P_i is conducted in a defined area. However, this inspection is carried out, at most, once.

$$if \ ABool_a; \ then \ AD_i = AD_p; \ else \ AD_i = 0 \tag{3.1}$$

Note that the properties in the plan are broken up by travel time to local amenities. This is covered in detail in Appendix C. Time windows in *PIRP* $[e_i, l_i]$ follow typical real estate inspections, which have a median duration of 30 mins. In *PIRP* the score of each property is a hedonic preference value from 1 to 4, this is defined in detail in Section 1.1. Finally like *OPTW*, *PIRP* consists of a single day plan, multi-day plans will be considered in future work. Note that these attributes are described in detail in Appendix A.

3.1.2 Constraints

The constraints on the *PIRP* problem that are consistent to the *OPTW* parent domain are:

- The plan starts at property P^{start} and finishes at property P^{end}.

- Each property $i \in P(P^{start} \cup P^{end})$ is visited at most once.
- The inspection start time at property i is within the time window $[e_i, l_i]$

Note that the *OPTW* constraint, time budget is limited to T^{max} is not valid in *PIRP*.

The *PIRP* domain constraints in addition to *OPTW* , are:

- The inspection finish time at property i is within the time window $[e_i, l_i]$
- The area inspection AD_i is completed prior to commencing travel to the next property. Note that this can be completed after inspection window closure l_i.
- The inspection for the last property P_n must finish prior to the overall finish time of the plan FT_p.
- If travel time is within 10 minutes walking, then walking will be used instead of the default plan travel mode.
- A travel mode will be selected and used for the entirety of a route, in Chapter 3 we cover this in detail.
- Final travel times are calculated using a directions API.

3.2 Assumptions

In this section we outline the basis of the assumptions we made when constructing the *PIRP* problem and its solution algorithms *PIRP-ILS* and *PIRP-C*.

Inspection Duration

We expect users to select an inspection duration of between 10 to 20 minutes. This is based on a survey of Melbourne and Geelong open homes collected on 17 Nov 21, for inspections during the period 18 Nov 21 to 22 Nov 21. The total number of inspections tallied was 10,827 over this period. These inspections showed a median open home duration of 30 minutes and a mean open home duration of 25 minutes. Unfortunately, publicly available data on the amount of time individuals typically spend inspecting is not available. So we assume that inspectors will (on average) plan on spending less than the average open time when inspecting a given property. Therefore, we set the test scenario inspection durations to 10 minutes, 15 minutes and 20 minutes. In future work, durations will be user editable.

Contract Grouping

Rent and buy searches are completely separated due to the low likelihood of cross-over of searches. Demographics determine whether a person is likely seeking a rental or purchase, with older demographics more likely to buy rather than rent, and vice versa, see [6].

Geographic Grouping

Zones In the dataset survey above, geographic zones are defined in the data source (realestate.com.au), this was used to break down the data into categories, in addition, two categories are defined by us. These have been broken up to reflect the different demographics present in the City, Inner Suburban, Suburban and Regional locations, see [47]. The pre-defined categories are, *Melbourne western suburbs*, *Melbourne northern suburbs*, *Melbourne eastern suburbs* and *Geelong - greater region.* The two categories defined by us are *city* (which includes suburbs of *Melbourne*, *Southbank* and *Docklands*) and *inner suburban* (which incorporates all suburbs within 10km of the CBD excluding the *City* category). Some Geographic zones overlap, however this only sees a 3.3% rate of duplication, with 3.13% of the overlap due to the Inner suburban category. For a dataset of 10,827 inspections, this will not impact significance in results.

Local government area There were 2,000 inspections for homes on sale in the northern suburbs of Melbourne on 20 Nov 21. It is unlikely any individual will consider a geographic catchment so large. For example, attending an inspection at 95 Goldmans Road, Cottles Bridge then attending an inspection at 25/122 Maribyrnong Road, Moonee Ponds (both in the northern suburbs on 20 Nov 21) would require a 50 minute (50.9 km) drive, indicating these two inspections should not be in the same category. To further break up the geographic zones we use local government area (LGA), this reduces the 2000 inspections in the northern suburbs on 20 Nov 21 into the following groups in table 3.1.

Table 3.1: Inspections to buy in the northern suburbs categorised by LGA 20 Nov 21

LGA	No. Inspections
City of Moonee Valley	240
City of Darebin	486
City of Hume	324
City of Banyule	231
City of Moreland	379
Shire of Nillumbik	62
City of Whittlesea	275
City of Brimbank	3

Minimum inspections The further categorisation above, makes some of these new categories no longer viable to inspect. We have assumed on a given day, in an LGA, if there are less than 10 available inspections, that this inspection category is not viable for our calculation. This is because people can easily construct a manual plan for a dataset this size. This means those properties are now excluded from comparison. Overall this excludes a total of 171 inspections, which equates to 1.59% of all inspections. This reduction has been considered minor, given the benefit of a more realistic inspection zone. This reduces the total Inspections used in the analysis to 10,592. An example of a category no longer viable is *City of Brimbank* above in Table 3.1, with only 3 inspections available.

Travel mode Individuals will restrict their search to specific geographic areas. For example, CBD locations have excellent walkability and public transport but smaller dwelling size, inner suburban locations have good walkability and public transport and larger dwelling size and outer suburban locations have poorer walkability and public transport and much larger dwelling size, see [23] and [24]. In samples using our primary dataset (*Victoria Inspections2.xlsx*) car speeds averages are CBD 15km/h, inner suburban 29km/h, suburban 44km/h and regional 49km/h. Conversely public transport speeds for a sample 2km journey are CBD 8.6km/h, inner suburban 7.1 km/h, suburban 2.2km/h and regional 5.5km/h, based on wait times and public transport distance to door. These factors influence the dominant mode of transport for a region. For this reason, for CBD properties, the public transport travel mode has been selected. While there is likely a mixture of public transport and car modes in inner suburban locations, car mode was selected for this, outer suburban and regional areas for the experiment.

Acceptable runtime

We assume users will accept a wait time, so long as it is less than 1 minute, which is equivalent to current *ILS* algorithm calculation time. Unfortunately in the domain of the web, acceptable waiting times have not been studied since 2004, see [36]. Nah investigated web page loading time, where dominant approaches meant loading time was noticeable for users. Give this lack of research, we base the assumption off the runtime of the best of breed *ILS* algorithm calculation time, and that runtimes in excess of this will not be acceptable.

Local amenity

Area inspection We assume that an *area inspection* is only required once for a general locality (such as a suburb). This concept represents the desire of an inspector to get to know the local amenities of the place they are looking to inhabit. However what makes a contiguous locality is subjective based on the favoured amenities of the person considering the locality. Examples of favoured amenities include public transport stations, supermarkets, restaurants, pubs or retail shops.

Travel to local amenity We assume that the travel modes popular to a region will govern the acceptable travel time, and travel mode. As discussed in Section 3.2, the suburban, regional and inner suburban mode of travel will be car and in the CBD public transit will be the chosen travel mode. Regional and suburban settings will have a travel time cap of 10 minutes to the nearest amenity, whereas inner suburban and CBD will be 5 minutes. These time-frames have been chosen because of the increased density of amenity towards the CBD. This has been noted as a feature of Melbourne's amenity by [40].

Time inspecting local amenity We will assume an average time of 10 minutes in inner suburban and CBD areas, and 20 minutes in suburban and regional settings, this will contain the maximum travel time to and return from the nearest amenity centre. Note that for the purposes of route planning, it is assumed the area inspection will be fully contained in the allowed for time, it will not become an additional step in the routing, because doing so would introduce complexity. However, this will be considered in future work. Further, we assume the area inspection can happen before or after (or be split before and after) an inspection, as browsing local amenity is not restricted by time windows.

Amenity chosen Individuals seek different amenities (e.g. restaurants, public transport, healthcare and schooling). Given this variety, groceries stores have been chosen as the amenity, with 91% of people buying groceries more than once a month, see [53], this is the most ubiquitous of the above amenities. In future work, we will allow users to define the amenity most important to them. This is covered in more detail in Appendix B.

3.3 Differences in PIRP algorithms

In this section we will initially outline the changes in the *PIRP* algorithms, then the potential advantages of both solutions, then the potential strengths of the *PIRP-ILS* solution. After this, the potential strengths of the *PIRP-C* solution will be discussed, following this, the high level approach of *PIRP-C* will be detailed with terminology.

Replace T_{max} Both algorithms remove T_{max}, which is used to limit the duration of a plan. A finish time is employed by both algorithms instead.

Remove Threshold *ILS* uses THRESHOLD run limit within ACCEPTANCECRITERION and also uses TIMELIMIT to limit the algorithm calculation time. The longer runtime of each comparison (due to the directions API), means that THRESHOLD is rarely reached. Therefore, THRESHOLD has been removed in the *PIRP* algorithms for simplicity.

Additional features *Area inspections* and *API directions* for travel time were included as new features. *API directions* had a significant effect on runtime, as seen in Section 5.9.2, which compares *PIRP-ILS* algorithm with and without the directions API. The runtime of each step when using API directions, can take more than 10 times the runtime than without. *API directions* allows for more accurate travel times, and multiple transport modes, while *area inspections* allow time for a common

Limit API calls Limit the impact of the use of directions API on calculation time, particularly as this operation in the algorithm described by [20] operates in a n^2 time complexity. This includes using euclidean distance with average speed calculations to remove unviable inspections, and only execute the API on viable inspections. Even with an 11% discrepancy between euclidean and directions API travel time, see [3], this will reduce unviable inspections. This saves time, so long as on average, more than 10% of properties are unviable. Further the monetary cost of each API call is reduced.

Limit API calculations to top 5 ratios In addition to removing unviable inspections, euclidean time calculations are used as the first step of creating a score/time ratio for viable property combinations. Only the top 5 ratios are selected for time re-calculation using the directions API. While an euclidean time calculation has inaccuracies, this allows us to reduce each step of API calculation comparisons from roughly 752 to 63 for *PIRP-C* and 317 to 25 for *PIRP-ILS*. Based on an average of 66 starting properties per dataset and an average output set of properties of 13 for *PIRP-C* and 5 for *PIRP-ILS*.

User customisation While the goal of this Minor Thesis is to provide a score maximising algorithm for the *PIRP* domain, research into scoring methodologies and underlying preferences is sparse. Given we are uncertain how accurate a high score closely matches a user's preferences, visual options for altering the route will be provided in both *PIRP* algorithms.

Score comparison Gunawan et. al. claims that by squaring the score (u) value, it improves algorithm performance. Given *PIRP* is a different domain than *OPTW*, we explore whether increasing (u^2), decreasing ($\sqrt{u}$) or leaving unchanged (u) is the ideal approach for *PIRP* algorithms.

Walking if short distance WTT_{ij} (walking travel time) is routinely calculated, using the same two-step viability-ratio approach. If walking time is less than 10 minutes then walking time is selected instead of the default transport mode. Otherwise, if invalid, then the default mode will be used (public transport or car travel time).

Change timelimit to 10 seconds Our experiments indicate that the TimeLimit value for *ILS* is set to 60 seconds. Preliminary analysis indicated 60 seconds worth of comparisons for *PIRP-ILS* would cost between \$0.60 and \$0.80 per algorithm run, while *PIRP-C* would cost \$0.05 to \$0.08 cost per algorithm run. Final comparisons are presented in 5.9.2. The following reasons contributed to reducing TimeLimit to 10 seconds:

- the budget for API experimental runs would be exhausted before the whole dataset was complete
- the *ILS* algorithm was not cost-competitive with the *PIRP-C* algorithm in preliminary runs
- lowering TimeLimit would not cut off the algorithm in a partially complete state, the TimeLimit constraint is only checked at the end of each *Improve* step.

This TimeLimit change was made to both *PIRP-ILS* and *PIRP-C*, to avoid giving an unfair advantage to *PIRP-C* by allowing it more time to resolve its answer.

3.4 Altered Algorithm - PIRP-ILS

In this section we describe the operation of *PIRP-ILS*, the algorithm adjusted from the best in class *OPTW* algorithm, called *ILS*. We then indicate the potential benefits of using this adjusted algorithm to address the *PIRP* problem. Note, alterations are indicated by blue text. Like *ILS*, *PIRP-ILS* consists of two overall phases. First the algorithm selects an initial solution (a set of scheduled properties) in a phase called ALTERED CONSTRUCTION then progressively replaces the properties in the solution with more optimal properties in the IMPROVE phase. This algorithm will potentially address *RQ2*, as it may solve the *PIRP* domain.

Construction Phase In *ILS*, CONSTRUCTION is made up of iteratively selecting viable insertion properties by comparing against all other viable properties. This does not lead to excessive runtime because travel times are based on simple distance-based calculations. In *PIRP-ILS*, the same euclidean time estimates from *ILS* are incorporated into viability, then on the smaller viable pool, conducts these calculations again using a directions API to provide final travel time calculations. As with *ILS*, the ratio score and travel time for each property are inputted into a ROULETTEWHEEL calculation to select a scheduled inspection. This is conducted iteratively until viable remaining unscheduled inspections are exhausted or the finish time exceeds the day finish.

Improve Phase - local search and perturbation We refer to PERTURBATION and LOCALSEARCH collectively as the IMPROVE phase. In LOCALSEARCH the four steps of SWAP, 2-OPT, INSERT and REPLACE are still conducted. For each step the same approach as *ILS* is conducted, however the two step viability and detailed calculation actions are used. Unscheduled properties are sorted by descending score value, and the highest score is compared first for INSERT and REPLACE steps. There is no change in the PERTURBATION algorithm. The full algorithm is below:

3.4.1 Potential advantages of PIRP-ILS

ILS offers an approach that has proven to be best in class for solving routing problems in the *OPTW* domain, which we consider as the parent domain to *PIRP*, see [20]. As covered in Section 2.7.1, algorithms that address the *OPTW* domain, should mostly address *PIRP*. While the literature demonstrates algorithms have been formulated to produce improved accuracy over naive approaches, they use simple approaches to calculate transport times (e.g. euclidean distance and average speeds, see [20]), and do not allow for multiple travel modes or inspection types.

We have indicated in Section 2.7.2 that euclidean distance-time does not provide sufficient accuracy for use in the *PIRP* domain. Therefore the use of the google maps directions API for travel time estimation for evaluation during the construction and improvement phases, is an important addition for *PIRP-ILS*. *PIRP-ILS* is promising because it addresses the additional concerns of *PIRP* within the framework of a demonstrated algorithm, proven in a parent domain.

Algorithm 5 ILS (P,R)

```
1: $S_0 \leftarrow$ CONSTRUCTION$(P, R)$
2: $S_0 \leftarrow$ LOCALSEARCH$(S_0, P*, P', R)$
3: $S* \leftarrow S_0$
4: NOIMPR $\leftarrow 0$
5: while TIMELIMIT has not been reached do
6:     $S_0 \leftarrow$ PERTURBATION$(S_0, P*, P')$
7:     $S_0 \leftarrow$ LOCALSEARCH$(S_0, P*, P')$
8:     if $S_0$ better than $S*$ then
9:         $S* \leftarrow S_0$                          ▷ *new* solution becomes *best* solution
10:        ~~NOIMPR $\leftarrow$ 0~~                    ▷ ~~*NoImpr* becomes 0~~
11:    else
12:        ~~NOIMPR $\leftarrow$ NOIMPR +1~~            ▷ ~~iterate *NoImpr*~~
13:    end if
14:    if ~~(NOIMPR+1) MOD THRESHOLD = 0~~ then         ▷ ~~*threshold* no. of iterations~~
15:        ~~$S_0 \leftarrow S*$~~
16:    end if
17: end while
18: return $S*$
```

3.5 New Algorithm - PIRP-C

Firstly we provide a high level overview of the differences in the *PIRP-C* algorithm, then we define the terminology of *PIRP-C* then outline the algorithmic approach, then finally outline the reasons that *PIRP-C* is likely to be good at addressing the *PIRP* problem. This algorithm will potentially address *RQ2*, as it may solve the *PIRP* domain better than *PIRP-ILS*.

3.5.1 PIRP-C Overview

PIRP-C is a novel algorithm, constructed with the *ILS* approach in mind, to determine if the *PIRP* domain is better served by a novel algorithm rather than an adapted mature *OPTW* algorithm.

The key differences in approach in the *PIRP-C* from the *PIRP-ILS* algorithm are:

- Construction phase - compare only the next 5 (sorted by late start) inspections. This reduces construction phase comparisons from polynomial (n^2) to a quasilinear ($n * log_n$) time complexity.
- Construction phase - select highest score property (no roulette wheel selection)
- Improvement phase - remove PERTURBATION function
- Improvement phase - remove LOCALSEARCH function, replaced with comparison of the lowest value scheduled properties, and highest value unscheduled properties (in a new approach called NEIGHBOURHOODCOMPARE).

3.5.2 PIRP-C Terminology

We define key terms used in the *PIRP-C* approach. The inspector will not view the property for a *Inspection Duration* Dur_p less than the entire *Inspection Window*. In the algorithm setup phase for the plan, the *Early Start, Early Finish, Late Start and Late Finish* (these terms originate from critical path analysis, see [26] and illustrated in Figure 3.1) are calculated using Dur_p:

- *Early Start* ES_i is simply equal to the *Inspection Start* e_i, in the *yellow box.*
- *Early Finish* EF_i is the *Inspection Start* plus *Inspection Duration*, $EF_i = ES_i + Dur_p$, in the *yellow box.*
- *Late Finish* LF_i is simply equal to the *Inspection Finish* l_i, in the *purple box.*
- *Late Start* LS_i is the *Late Finish* minus *Inspection Duration*, $LS_i = LF_i - Dur_p$, in the *black text.*
- *Scheduled Start* S_i^{Plan} is the planned start time. It can occur in a range between the *Early Start* and *Late Start* $ES_i \leq S_i^{Plan} \leq LS_i$, in the *orange box.*
- *Scheduled Finish* can occur in a range between the *Early Finish* and *Late Finish.* $EF_i \leq F_i^{Plan} \leq LF_i$. It is also $F_i^{Plan} = S_i^{Plan} + Dur_p$, in the *orange box.*

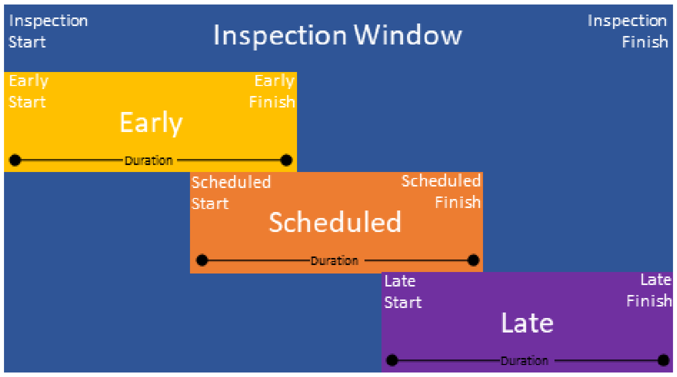


Figure 3.1: Inspection Window

Consider the scenario presented in Figure 3.2: Previously scheduled inspection (h) and five unscheduled inspections ($i_1, ..., i_5$), our aim is to add a combined maximum score *scheduled inspection* sequence to our plan. The red dashed line illustrates the scheduled finish (F_h^{Plan}) of inspection h, the *green* boxes contain valid start times after F_h^{Plan}, and *red* indicates invalid start times before F_h^{Plan}. If the late time-frames are valid, then the *unscheduled inspection* (indicated as *blue*) remains valid. If the late time-frame is invalid, the *unscheduled inspection* becomes *grey*, to show the inspection is now invalid.

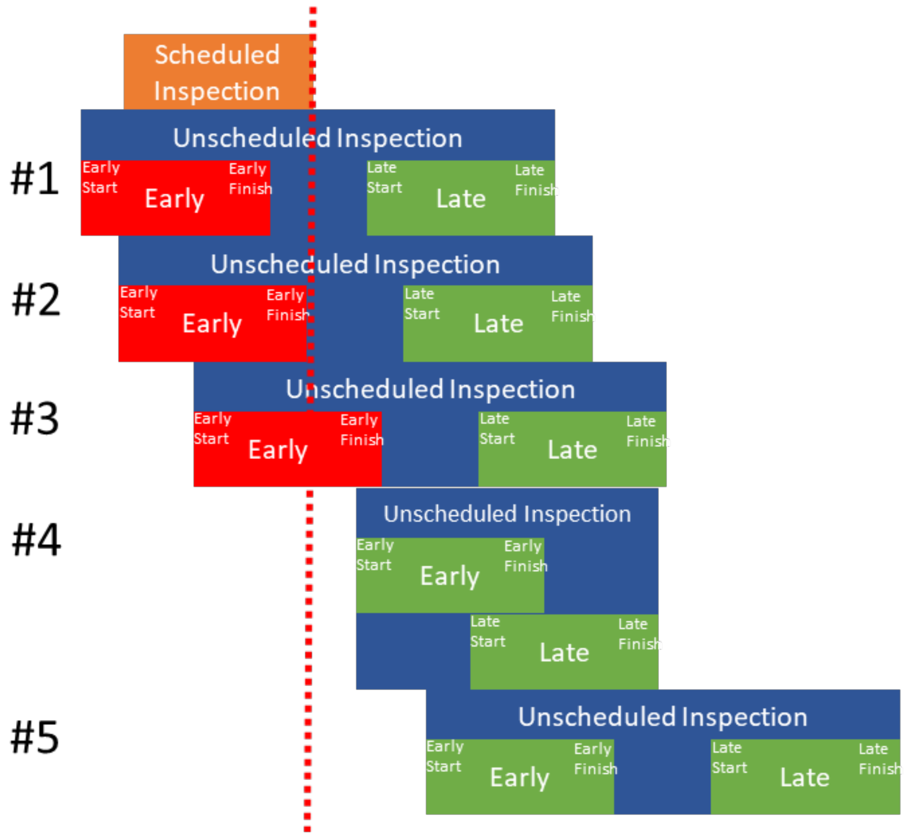


Figure 3.2: Sorted Inspections

In Figure 3.2, the early start times of inspections i_1, i_2 and i_3 are before the scheduled finish of h, i.e., $ES_{i_1} < F_h^{Plan}$, $ES_{i_2} < F_h^{Plan}$ and $ES_{i_3} < F_h^{Plan}$. Therefore, if we use the early start to determine validity, then only i_4 and i_5 would be considered viable. However, by considering late start start times given $LS_i > F_h^{Plan}$, all inspections are still viable. *Late start* is useful when some overlap in timings with the previous inspection occurs. However *early start* is useful when there is no overlap, this allows more inspections to fit within the plan. Where partial overlap exists, the earliest viable time between *early start* and *late start* is selected. To capture all viable inspections we sort unscheduled inspections by *late start* and compare early and late start to create the densest practical list of scheduled inspections.

The next element in assessing viability is to consider travel time. Travel time requires the locations of the source inspection h and destination unscheduled inspections $i_1, ..., i_5$. Additionally for directions API travel time calculation, the finish time of the scheduled inspection is required to calculate traffic delay. In Figure 3.3 each rounded rectangle TT represents the *travel time* from F_h^{Plan} to LS_i. We can see that prospective unscheduled inspection i_1 and i_3 have become unviable (now grey) as a result of their travel times:

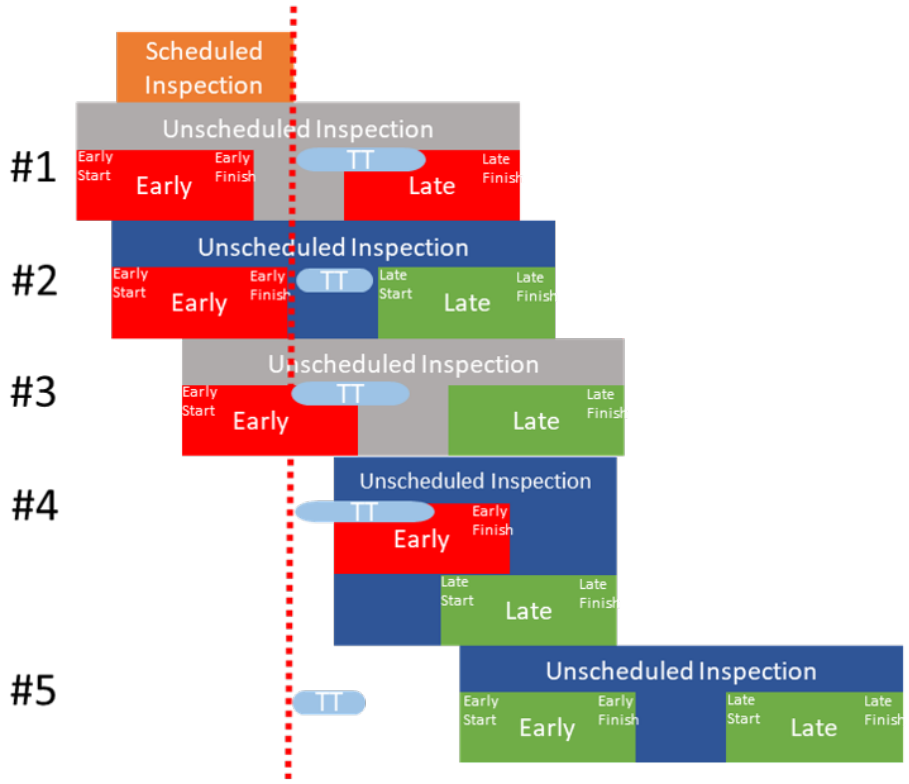


Figure 3.3: Sorted Inspections with Travel Time

After *travel time* is considered, the *area inspection* variable is calculated for each inspection, *area inspections* are introduced in Section 3.1.1. In Figure 3.4 i_2 has an area inspection satisfied by a previous scheduled inspection. This is denoted by the *clear cylinder with dark blue outline*, and means this *area inspection* does not need to occur. Unscheduled inspection i_4 and i_5 have not been previously satisfied (and therefore need an *area inspection*), they are denoted by a *filled dark blue cylinder*. N.B an *area inspection* can occur, before (e.g. i_5), after, or be split to occur before and after the inspection (e.g. i_4). The duration of the area inspection is added to the scheduled inspection duration (e.g. i_4 and i_5). In i_5 the gap between the *area inspection* and *travel time* indicates waiting time:

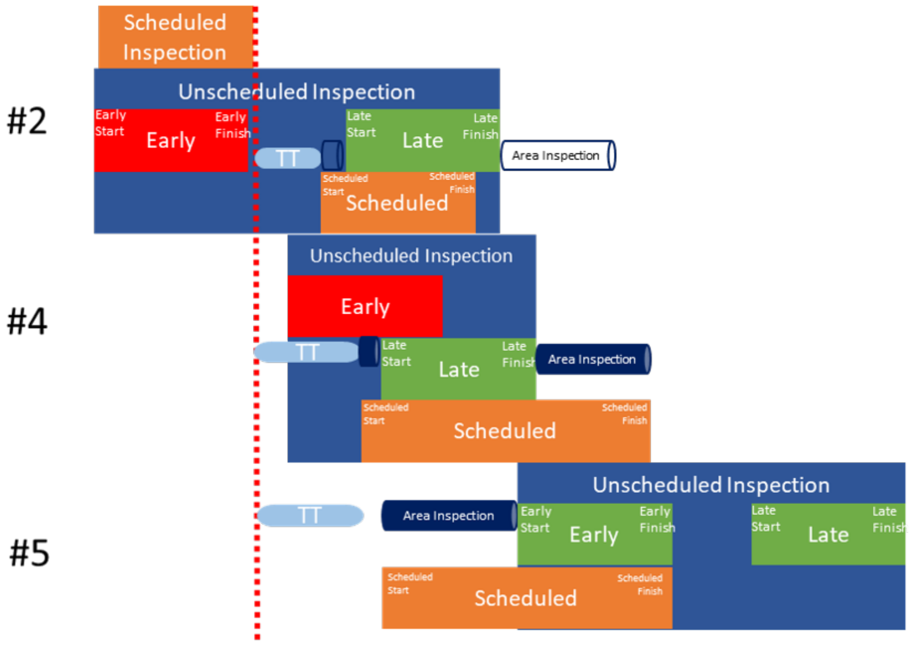


Figure 3.4: Sorted Inspections with Travel Time & Area Inspections

3.5.3 PIRP-C Approach

Our algorithm *PIRP-C* is made up of two steps in Algorithm 6. The first step, *Construct* is made up of TRIM and ALTERED CONSTRUCTION and the second step *Improve* is made up of INSERTC and NEIGHNOURHOODREPLACE.

In our implementation of LOCALSEARCH both the SWAP and 2-OPT procedure are removed. A sample output for *PIRP-C* was examined *10MinInspection20211120-West-Maribyrnong-Buy1OutputPirpC.txt* to evaluate how often the SWAP or 2-OPT would successfully execute. This is one of the more favourable datasets for this comparison, because the geographic area it encompasses is relatively small (therefore travel times are short), and the inspection time was set to only 10 minutes. The dataset had 18 scheduled properties. We found that two pairs could be reordered. In both cases these were neighbouring inspections, therefore SWAP was found to be wasteful, as these would both more efficiently be matched by 2-OPT. Further, we found that reordering these inspections did not result in a beneficial reduction in overall travel time. Therefore, in this example 2-OPT did not add value. The reduced effectiveness of these two steps is due to the narrower time windows, typical of *PIRP*. If we look at the original dataset *20211120-West-Maribyrnong-Buy1.csv* the average inspection window duration is 29.7 minutes long, therefore these two procedures provide less value for each comparison they conduct than in the *OPTW* domain. For this reason, they are removed from *PIRP-C*.

In Algorithm 7 the start/finish property is added to the scheduled inspection plan and removed from the unscheduled inspection list. Each inspection is pre-sorted by their late start LS_i which is $LS_i = LF_i - Dur_i$. While entering a new inspection continues to return valid unscheduled insertion inspections, iteratively inspections are added to the solution. The first

Algorithm 6 PIRP-C

```
1: $P \leftarrow$ TRIM($P$)    ▷ remove inspections start/finish outside day start/finish
2: $S_0 \leftarrow$ ALTERED CONSTRUCTION($P^*, P'$)    ▷ Build initial solution
3: $S_0 \leftarrow$ INSERTC($S_0, P^*, P'$)    ▷ Add high value inspections in gaps
4: $S_0 \leftarrow$ NEIGHBOURHOODREPLACE($S_0, P^*, P'$)    ▷ Replace with high value inspection
5: return $S_0$
```

Algorithm 7 ALTERED CONSTRUCTION(P^*, P')

```
1: $P^* \leftarrow P^{start}$ and $P^{end}$    ▷ scheduled properties
2: $P' \leftarrow P \setminus P^{start}$ and $P^{end}$    ▷ unscheduled properties
3: sort all $p \in P'$ by $LS_i$    ▷ sort unscheduled insp. by late start
4: Initialise $S_0 \leftarrow P^*$
5: while $Comp(i)! = 0$ do    ▷ Until no competitors are returned
6:     $i \leftarrow$ $FindStart(P', p)$    ▷ get first late start after scheduled finish
7:     Calculate $Comp(i)$    ▷ Competitors, tested for viability
8:     $S^* \leftarrow Comp(i)$    ▷ Select highest scoring inspection in competitors
9:     $P' \leftarrow P' \setminus \{ p^* \}$    ▷ Remove from unscheduled properties
10: end while
11: return $S_0$
```

step of each iteration is to search for the first unscheduled property, whose late start is after the current scheduled inspection finish time. This means that the next algorithm does not need to consider unscheduled inspections whose late start is before the current scheduled finish time, who by definition are unviable. Next, a window of the next 10 inspections are used, each are tested for viability, then their score/travel time ratios are calculated using the euclidean approach. The top 5 of these are then have their viability and ratios recalculated using the directions API. Finally inspection with the highest ratio is added to the inspection plan. The main differences in approach from *PIRP-ILS* is that sorted inspections are used to limit comparisons to 10 competing inspections each iteration, whereas *PIRP-ILS* must compare all remaining inspections against each other. This is particularly wasteful in the *PIRP* domain, with narrower time windows than *OPTW*, and where are more deterministic approach is realistic. The *C* in the overall algorithm name *PIRP-C* is derived from this approach, *C* stands for *competitors* where a small number of inspections must directly compete to win selection. The viable competing solutions to ALTERED CONSTRUCTION are saved and presented to the user in the results as an optional selection, as discussed in the *User Customisation* paragraph of Section 3.3.

In Algorithm 8, the top 30% of unscheduled inspections are tested for insertion into the inspection plan. To successfully insert, the inspection must not impact the neighbouring inspections. First, the last inspection that has a finish time before the start time of p is defined as i. The following scheduled inspection is defined as j, then if p can be inserted between i and j without changing their start and finish times, then p is inserted into the inspection plan at this location.

The entirety of Algorithm 9 is unique. Initially as with Algorithm 8 the top 30% of unscheduled inspections are considered. However, instead of inserting, the approach here is to replace

Algorithm 8 InsertC(S_0, P^*, P')

```
1: Sort all $p \in P'$ desc. order on $p_u$          ▷ unscheduled insp. largest score first
2: for $p \in P' <$ UThreshold do
3:     $i \leftarrow$ LastOccurance($P^*$)            ▷ the inspection to insert p after
4:     $j \leftarrow$ LastOccurance($P^*$)+1          ▷ the inspection to insert p before
5:     if $S_p < F_i \wedge F_p < S_j$ then
6:         $P^* \leftarrow p$                          ▷ add p
7:     end if
8: end for
9: return $S_0$
```

Algorithm 9 NeighbourhoodReplace(S_0, P^*, P')

```
1: for $p \in P' <$ UThreshold do
2:     $x \leftarrow p$                                ▷ unscheduled inspection
3:     $pr \leftarrow$Select($i, P^*$)                 ▷ find index of replacement
4:     $i \leftarrow$Select($i, P^*$)+1                ▷ find previous inspection
5:     $prPr \leftarrow$Select($i, P^*$)-1             ▷ find 2 previous inspection
6:     $n \leftarrow$Select($i, P^*$)+2                ▷ find next inspection
7:     $nN \leftarrow$Select($i, P^*$)+3               ▷ find 2 next inspection
8:     if $x_u > i_u \wedge F_{pr}^{Plan} < S_x \wedge S_n^{Plan} > F_x$ then      ▷ Step 1
9:         $P^* \leftarrow x$                          ▷ Add x to scheduled insp.
10:        $P' \leftarrow i$                           ▷ Remove i from schehduled insp.
11:    else
12:        if $x_u > (i_u + pr_u) \wedge F_{prPr}^{Plan} < S_x \wedge S_n^{Plan} > F_x$ then      ▷ Step 2
13:            $P^* \leftarrow x$                      ▷ Add x to scheduled insp.
14:            $P' \leftarrow i$                       ▷ Remove i from schehduled insp.
15:            $P' \leftarrow pr$                      ▷ Remove pr from schehduled insp.
16:        else
17:            if $x_u > (i_u + n_u) \wedge F_{pr}^{Plan} < S_x \wedge S_{nN}^{Plan} > F_x$ then      ▷ Step 3
18:                $P^* \leftarrow x$                  ▷ Add x to scheduled insp.
19:                $P' \leftarrow i$                   ▷ Remove i from schehduled insp.
20:                $P' \leftarrow n$                   ▷ Remove n from schehduled insp.
21:            else
22:                if $x_u > (i_u + pr_u + n_u) \wedge F_{prPr}^{Plan} < S_x \wedge S_{nN}^{Plan} > F_x$ then      ▷ Step 4
23:                    $P^* \leftarrow x$              ▷ Add x to scheduled insp.
24:                    $P' \leftarrow i$               ▷ Remove i from schehduled insp.
25:                    $P' \leftarrow pr$              ▷ Remove pr from schehduled insp.
26:                    $P' \leftarrow n$               ▷ Remove n from schehduled insp.
27:                end if
28:            end if
29:        end if
30:    end if
31: end for
32: return $S_0$
```

one or more inspections, so that the plan has a higher overall score. Similar to Algorithm 8, a search algorithm is used to locate an insertion point. Here pr (for previous) is the inspection found first, this inspection is the last scheduled inspection whose finish is before the late start of inspection p. Then the next inspection i (the most likely replacement) is initialised, similarly,the following inspection is initialised as n (for next) and the following inspection from n is initialised as nN (for next, next). The previous inspection to pr is initialised as $prPr$ (for previous, previous). Each of these variables are used for the steps 1-4. All steps have a similar approach, where their bounding inspections are evaluated to see if any change to their start or finish time occur (as in Algorithm 8), and if this does not occur, then this/these inspection(s) are removed, and p (x later in the algorithm) is inserted into the plan. The potentially removed inspections for each step are:

- Step 1 - i
- Step 2 - i and pr
- Step 3 - i and n
- Step 4 - i, pr and n

These steps are illustrated in in Figure 3.5.

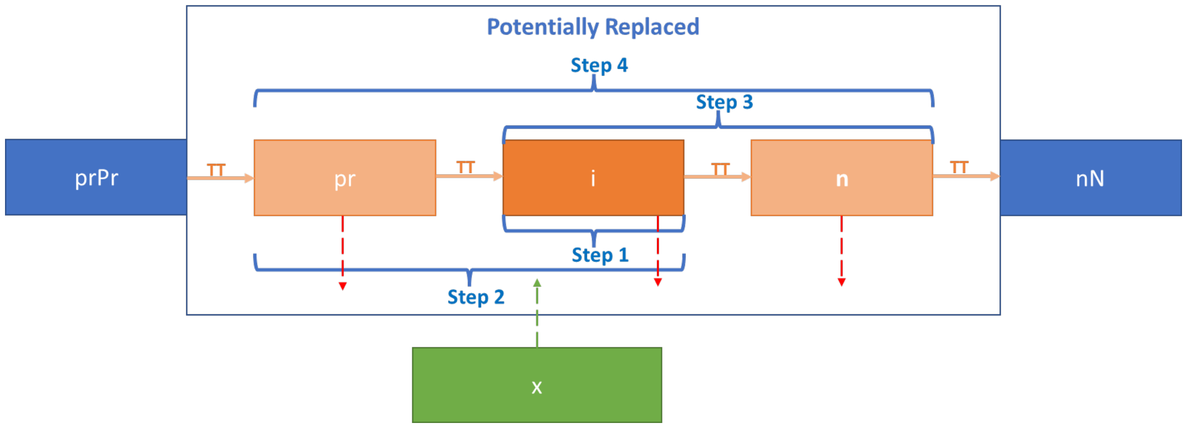


Figure 3.5: Neighbourhood Replace Steps

NeighbourhoodReplace and InsertC replace the Perturbation and LocalSearch functions for *PIRP-C*. They are less random, and seek to take advantage of the more constrained time windows of *PIRP* by using a more deterministic (and less random) approaches to insertion and replacement.

3.5.4 Potential advantages of PIRP-C

It is likely wasteful to use an algorithm that uses several points of random operation when the new domain *PIRP* supports a more deterministic approach. Mathlouthi et. al. agrees, inidicating that an increase in the duration-taken:time-window ratio (as in *PIRP* compared with classic *OPTW*) reduces combinatorial complexity. Using a more deterministic approach would mean quicker operation of the algorithm, and if *PIRP-ILS* cannot resolve towards an optimal result quickly due to the slower operation of the direction API (compared with euclidean-only travel time calculation), a more deterministic approach would lead to a higher average score than *PIRP-ILS*.

In the dataset *Victoria Inspection.xlsx* the median inspection time is 30 minutes. With inspection durations of 10, 15 or 20 minutes (see Section 3.2) leads to less overlap in inspections than in the *OPTW* domain. This is illustrated below in Figure 3.6. The blue bars with the prefix *PIRP* represent the narrow time windows typical of *PIRP*, here we can see that few of the bars overlap. Conversely, typical *OPTW* use cases have wider time windows, representing opening hours of point of interest or service windows. This is also illustrated in Figure 3.6, the yellow bars have the prefix *OPTW*. Below is a naive construction (before travel time is considered and with a simplified 30 minute inspection time) for the *Construction* step of the algorithm for both the *OPTW and PIRP*:

Construct OPTW solution To construct the first step of the solution, *OPTW* must compare *OPTW-1, OPTW-2 and OPTW-4*. To construct the second step, *OPTW* compares *OPTW-1, OPTW-2, OPTW-3 and OPTW-4* (excluding inspection selected in step 1). To construct step 3, *OPTW* compares *OPTW-1, OPTW-3, OPTW-4 and OPTW-5* (excluding inspections selected in step 1 and 2). Finally, to construct step 4, compare *OPTW-1, OPTW-3, OPTW-4 and OPTW-5* (excluding inspections selected in step 1, 2 and 3).

Construct PIRP solution To construct the first step of the solution, *PIRP* has no comparison to conduct, *PIRP-1* is selected. To construct the second step, *PIRP* compares *PIRP-2 and PIRP-3*. To construct step 3, *PIRP-4* is selected. Finally, to construct step 4, *PIRP-5* is selected.

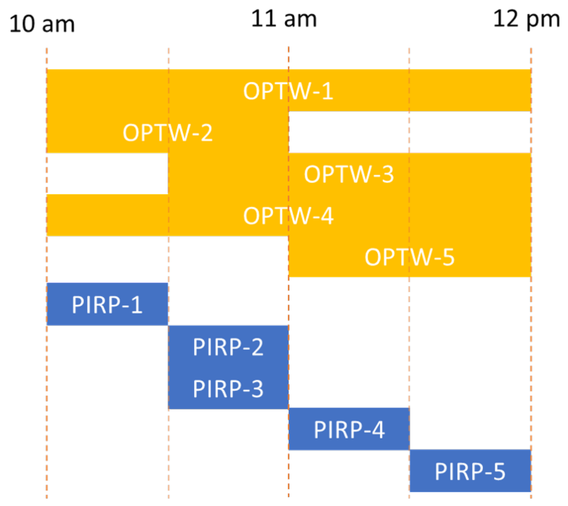


Figure 3.6: PIRP vs OPTW Time Window Width

3.6 Summary

In this chapter we formalised the *PIRP* problem, outlined our assumptions, highlighted the different approach both algorithms use from the *OPTW-based ILS* algorithm, and outlined the *PRIP-ILS* and *PIRP-C* algorithms in full with their potential benefits.

The formalisation highlighted the essential attributes of *area inspection* and the replacement of the max plan duration with a finish time for the plan.

The assumptions outlined that inspection durations are 10, 15 and 20 minutes, and highlighted factors driving geographic grouping and local amenity inspection (*area duration*).

The differences in the *PIRP* algorithms from the *OPTW-based ILS* algorithm highlighted that the key differences included introducing *API directions*, using methods to limit API calls, allowing user customisation, conducting walking if short distances. Further, TimeLimit was reduced from 60 to 10 seconds and we verified if our score sensitivity is correctly set by analysing several sensitivities. Further, the Threshold and T_{max} variables were been removed. All of these steps made both algorithms more fit for purpose for the *PIRP* domain. Comparing results with a higher/unconstrained TimeLimit will constitute future work, due to the potential impact limited TimeLimit has on *PIRP-ILS* results.

The *PIRP-ILS* algorithm Construction and Improve phases were described. Further we described the main benefit of *PIRP-ILS*, which is that it is minimally changed to and *ILS* is best in class for the parent domain of *PIRP*. Therefore, if the differences between the *PIRP* and *OPTW* domains do not signficantly effect the algorithms, then *PIRP-ILS* should perform better than *PIRP-C*.

And finally the *PIRP-C* algorithm was described, with the novel step of Neighbourhood Replace and adjusted insert approach referred to as InsertC, further comparisons of unscheduled properties are restricted through all steps of the algorithm. This is done by using a comparison window with the unscheduled inspections sorted by their *late start*, a concept described in detail in Section 3.5.3. We claim that the narrower time windows of *PIRP* than *OPTW* support a more deterministic algorithm, and suggest that *PIRP-C* will resolve to an answer quicker, based on this approach. This could, potentially lead to shorter runtimes and higher scores than *PIRP-ILS*.

Chapter 4

Experiment

This chapter introduces the experiments we conducted to analyse the proposed algorithms: we define the formal setting of our experiments and specify configuration of execution environment, to enable reproducibility of the experiments.

4.1 Defining the experiment

In this section, we outline the reasons for conducting a novel experiment, the success parameters, data features used, the source of data, approach to scoring and finally the exclusions from the algorithm.

Reasons for conducting the experiment Several *OPTW* algorithms test their solutions against modified data from the vehicle routing domain, for example [43] uses Solomon [46] and [20] uses both Solomon [46] and Cordeau et al. [12] test data sets. However, these data sets cannot be translated to the *PIRP* domain for the following reasons:

- *PIRP* data should be collected from real-world *MLS* inspections. In our case we use the Greater Melbourne and Geelong regions. Whereas Cordeau et al. [12] and Solomon [46] geographic locations are randomly dispersed in a grid, we believe a sample from real-world inspections is more valuable than a random distribution as there are geographic features that may be distinct to the *PIRP* problem.
- The addition of a local area inspection parameter changes the nature of the *time window* attributes in *OPTW* datasets, therefore the datasets cannot be accurately equated.

Success parameters The runtime and score are considered the success parameters for a successful *PIRP* algorithm. This is consistent with the parameters measured in Solomon et al., Cordeau et al., Righini et al. and Gunawan et al. see, [46], [12], [43] and [20].

Data features The features of algorithm stage, sensitivity, duration, number of inspections, mode and region are used in the results analysis. Algorithm stage was chosen because the initial stage of the measured algorithm may have a more potent effect on the score for the runtime taken. Therefore a stage-one only algorithm may be more attractive than the full algorithm. Sensitivity is covered in Section 3.3, where we posit that the new domain of *PIRP* deserves a re-examining of the score sensitivities. The multiple scenarios for inspection duration are covered in Section 3.2. The number of inspections is considered because the new algorithm *PIRP-C* was built to conduct less comparisons than the *PIRP-ILS* algorithm, so it is important to compare their operation in a variety of inspection dataset sizes. Inspection mode of *rent* and *buy* introduced in Section 3.2 are split because of the potential demographic differences. Region is one of the most important data features, where each region hosts different local amenities and a variety of density, road and public transport conditions.

Data collection Data was drawn from inspections listed on *realestate.com.au*, which was selected as a data source due to its leading market share of 58%, see [34]. Note that all data used in the experiment, is available on a public GitHub repository: https://github.com/ChrisBond-s3767838/PIRP-Minor-Thesis.

Scoring preference classes We split the preference classes into the following categories, following the approach from Mueller et al. [35]: the most important with a score of 4, potentially important with a score of 3, medium importance with a score of 2 and low importance with a score of 1. A weighted random approach to assigning these categories was undertaken where each category would be assigned to a percentage range of each inspection in a dataset. The most important category would be assigned to between 3%-10% of all inspections, the potentially important and medium importance categories would be assigned to between 5%-20% of all inspections and the low importance category would be assigned to between 40%-70% of all inspections. The breakdown was calculated per dataset and rounded up to the nearest inspection.

Exclusions The loading of dataset from disk and results output to txt files are not included in the calculation of either algorithm. It is considered that these input/output operations could vary greatly depending on the implementation platform, and do not feature as core aspects of the tested algorithm. Also, datasets were excluded if they were fewer than 10 records.

4.2 Define Execution Environment

A desktop PC was used for hosting algorithm inputs, execution and results, a Google Directions API was leveraged for travel time calculations.

4.2.1 Hardware

The system details of the desktop PC used for execution of the algorithm is below:

Table 4.1: Execution System Hardware Details

Category	Value
Processor	Intel Core i5-6600K CPU @ 3.50GHz 3.50 GHz
Installed RAM	16.0 GB
System type	64-bit operating system, x64-based processor
Hard Disk	NVME Samsung SSD 950

4.2.2 Software

Both algorithms (*PIRP-ILS* and *PIRP-C*) were defined in a Java application, executed within the Eclipse IDE 2021-12. A synchronous Google Maps Directions API call was made to define travel times. Google directions API was chosen given its roughly 80% market share, see [38]. Both algorithms shared the following algorithm elements:

- *CalculationsPackage* – Ratio calculation function, time calculation functions, function for Euclidean calculations for distance, function for the average speed of travel mode in specific region (used in Euclidean time calculations)
- *SortPackage* – QuickSort decreasing by value and sorting unscheduled inspections by late start.
- *APIrequest* – API request function and JSON parser to translate results.
- *InputOutput* – Functions to read, write and list the relevant files to be read in. e.g. File streaming was used to ingest datasets from csv and FileWriter is used to write out to txt files.
- *DataObjects* – Objects for file paths, file information, area inspection, inspection, property, travel type and scheduled inspection. These objects are actively updated through the course of input and algorithm execution, and the scheduled inspections are written out to the result files.
- *SharedOperations* – Function to remove unscheduled inspections that fall outside of the defined bounds of the day, function to update area inspection state, function to add scheduled inspection, function to find index of scheduled inspection, function to sort scheduled inspections by value, function to define a scheduled inspection and various getters and setters for data objects and overall variables (such as starting location and day start time).

The following parameters were used for the Google Maps directions API:

- *TravelMode* - Public transit with walking, car with walking and walking modes. Here public transit was used as the transit mode for CBD inspection datasets, whilst car was used for other location datasets, and walking for sub-10 minute walking journeys.
- *Originlocation* - A string address representing the origin address

- *Destinationlocation* - A string address representing the destination address
- *Departuretime* - algorithmically determined via the finish time of the origin location inspection.

4.3 Summary

In this chapter we introduced the experiment we conducted, including justifying that datasets in the *OPTW* domain are not applicable to *PIRP*, highlighting that acceptable runtime and optimised score will be the parameters for success. Further, we outlined the selected data features and collection approach, then highlighted the structure of a semi-randomised hedonic preference approach to scoring. Finally for the experiment we marked out that dataset input and result output operation calculations would not be included in the runtime measurement for the algorithm. Next, we defined the execution environment, providing the specifications of the desktop PC used to run the algorithm, then providing a breakdown of the software used, highlighting that Google Cloud Platform was used and the common algorithm elements of both *PIRP-ILS* and *PIRP-C*. Finally we describe the parameters entered into the Google Maps Directions API, so that this can be replicated if desired.

Chapter 5

Results and Evaluation

In this chapter we find that overall the *PIRP-C* algorithm performs better than *PIRP-ILS*, and performs better across all data features captured in our dataset.

Firstly we summarise the overall results, then compare against a variety of dataset features to analyse if any different trends are present in certain data features than in the summarised set. Next, we conduct an alternative analysis to understand the cost considerations and the impact of introducing API calculations has on each algorithm.

5.1 Approach

The results of *PIRP-ILS* and *PIRP-C* were analysed. The key outputs analysed were the time taken in seconds to perform the calculation and the overall score achieved by the algorithm. *PIRP-ILS* is the base case, whereas *PIRP-C* is the novel algorithm. It was not possible to conduct a comparison against classic best known solution values obtained by state-of-the-art algorithms in the literature, as these scenarios do not match the attributes of the *PIRP* domain. We find that in the *overall* scenario, the *PIRP-C* algorithm performs better for both the average and worst-case settings. Additionally, we compared the algorithms using each of the dataset features listed below:

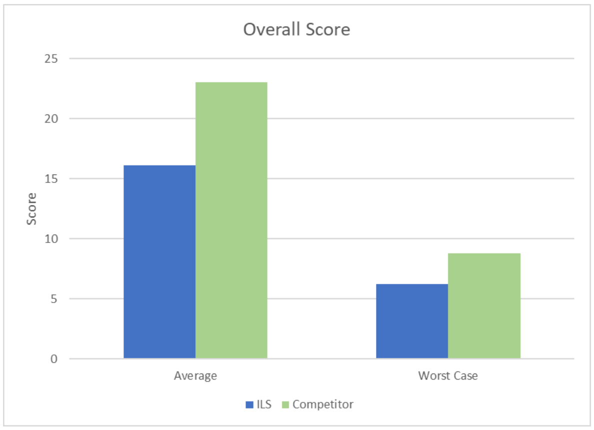
- Overall
- Algorithm stage
- Sensitivity
- Inspection duration
- Number of inspections
- Inspection mode
- Region

5.2 Overall Summary

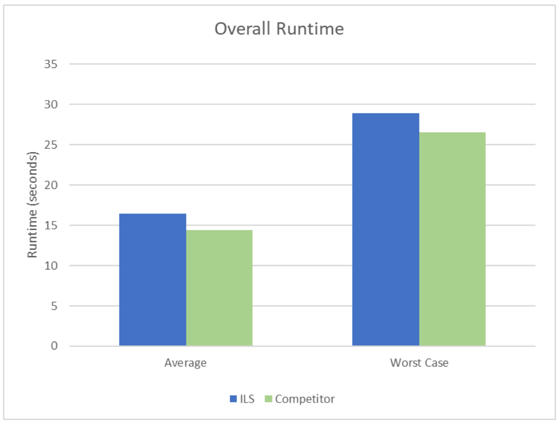
A total of 1,251 results for both algorithms were collected. This excluded 72 datasets that experienced dataset errors due to no records meeting the criteria of being within the specified daily time window of 9am-6pm. The time taken in seconds and the score achieved were captured for each algorithm for each run. Here the worst case was considered the first quartile in the score results and the third quartile in the runtime results. A summary of the 1,251 results is below:

Table 5.1: Overall Algorithm Results

Overall	Average Score	Score St Dev	Average Run Time	Run Time St Dev
PIRP-ILS	16.11	9.89	16.44	12.43
PIRP-C	23.03	14.27	14.37	12.19



(a) Score



(b) Runtime

Figure 5.1: Overall Comparison of Algorithms

What this indicates is that the *PIRP-C* (Competitor) algorithm gives a higher value in a quicker time, on average than the *PIRP-ILS* (ILS) algorithm when provided with *PIRP* datasets.

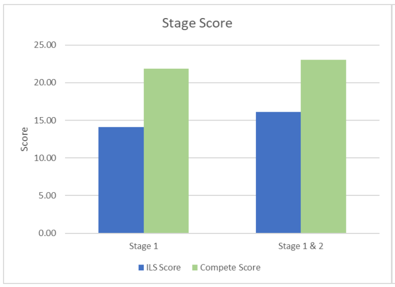
For the worst case, the *PIRP-C* score achieved is higher at 8.76 than *PIRP-ILS* at 6.22, for time taken *PIRP-C* is marginally lower at 26.56 seconds than *PIRP-ILS* at 28.88 seconds. Overall, the performance per second is better in *PIRP-C* with a score of 1.6 points/second vs 0.98 for *PIRP-ILS* in the average case and 0.33 for *PIRP-C* and 0.21 for *PIRP-ILS* in the worst-case scenario. Further, we find that *RQ3* is met by both *PIRP-ILS* and *PIRP-C*, as both algorithms have a runtime less than the in the *OPTW-based ILS* algorithm. Additionally, we find that *RQ2* is best met by *PIRP-C*, which offers shorter runtime and higher scores than *PIRP-ILS*.

5.3 Algorithm Stage

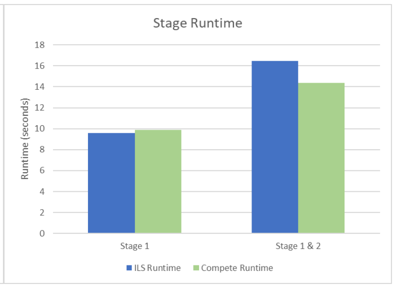
The algorithm details were checkpointed at the conclusion of stage 1 (construct) and then again at stage 2 (improve). The second stage is dependent on the first stage.

Table 5.2: Algorithm Stage Results

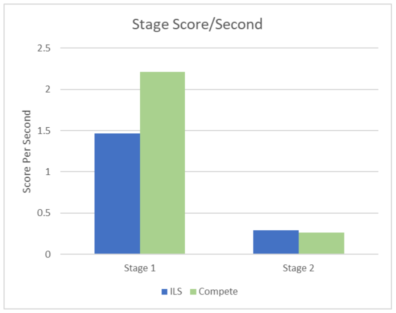
Overall	PIRP-ILS Score	PIRP-C Score	PIRP-ILS Run Time	PIRP-C Run Time
Stage 1	14.11	21.84	9.60	9.88
Stage 1 and 2	16.11	23.03	16.44	14.37



(a) Score



(b) Runtime



(c) Score/Runtime Fraction

Figure 5.2: Stage Comparison

The *Stage 1* run time is marginally shorter for *PIRP-ILS*, but the score is significantly lower. During *Stage 2*, a similar amount of time and score improvement is experienced by both algorithms. This indicates that there is no significant alternative trend than presented in section 5.2. However, given the relatively minor score improvement of stage 2 in *PIRP-C*, this invites future work to either reduce runtime or improve the score of stage 2, or both.

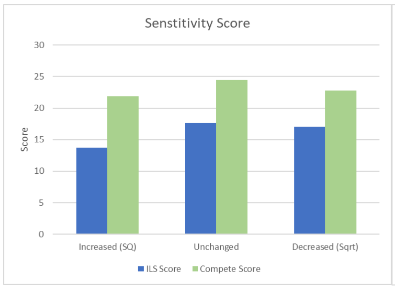
5.4 Sensitivity

Both algorithms were run multiple times using varying sensitivities, below:

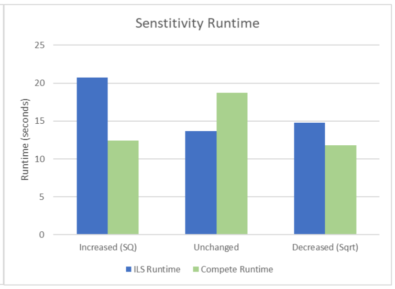
Table 5.3: Sensitivity Results

Overall	PIRP-ILS Score	PIRP-C Score	PIRP-ILS Runtime	PIRP-C Runtime
Increased (SQ)	13.75	20.87	20.75	12.40
Unchanged	17.62	24.45	13.65	18.75
Decreased (SQRT)	17.05	22.76	14.79	11.78

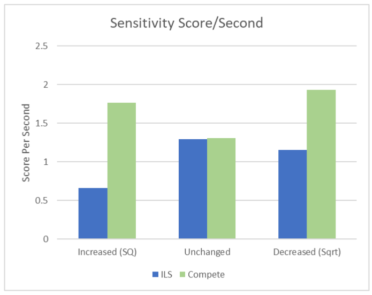
PIRP-C score and runtime is more optimal as the sensitivity is increased. Whereas *PIRP-ILS* performs most optimally where the score ratio sensitivity is unchanged. However, these variations are not sufficient to overcome superior *PIRP-C* performance. Run time seems to converge on 11.6 seconds in each algorithm's optimal configuration. No significant alternative trend is outlined in this section that challenges the conclusion presented in section 5.2.



(a) *Score*



(b) *Runtime*



(c) *Score/Runtime Fraction*

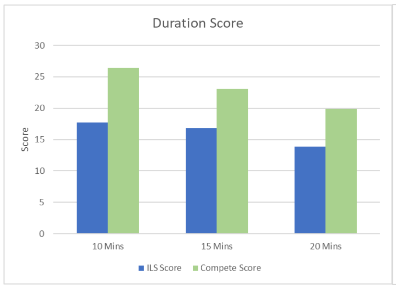
Figure 5.3: Sensitivity Comparison

5.5 Inspection Duration

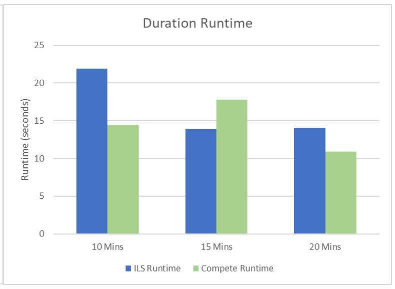
Each dataset was duplicated for the three inspection durations provided, below are the results:

Table 5.4: Inspection Duration Results

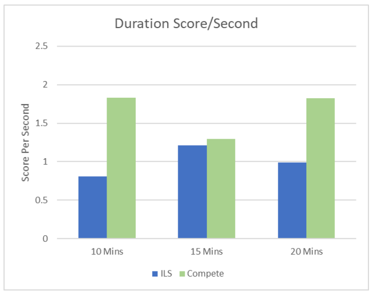
Overall	PIRP-ILS Score	PIRP-C Score	PIRP-ILS Runtime	PIRP-C Runtime
10 Mins	17.74	26.44	21.88	14.44
15 Mins	16.83	23.07	13.88	17.77
20 Mins	13.90	19.87	14.02	10.89



(a) *Score*



(b) *Runtime*



(c) *Score/Runtime Fraction*

Figure 5.4: Duration Comparison

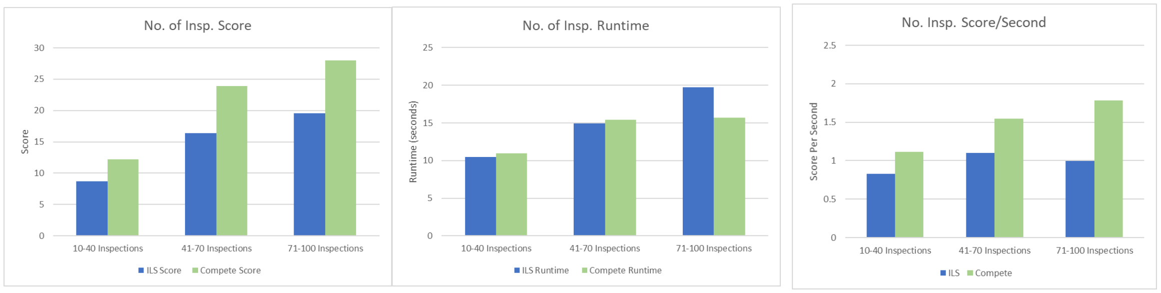
Both algorithms score best with shorter inspections, whereas the runtime for each varies significantly. Anecdotally, it is not surprising that a shorter inspection duration would lead to a higher score due to increased inspection density. Interestingly there is no consistent pattern for run time. *PIRP-ILS* performs marginally better for runtime in its optimal configuration of 15 minute inspections than *PIRP-C*'s 20 minute inspections. However, the trend of superior performance of the *PIRP-C* algorithm continues for this data feature.

5.6 Number of Inspections

Datasets were categorised by their size, to see if either algorithm performed better on different sized datasets: The trend of the score being improved by *PIRP-C* is consistent across all

Table 5.5: no. inspections results

Overall	PIRP-ILS Score	PIRP-C Score	PIRP-ILS Runtime	PIRP-C Runtime
10-40 Inspections	8.71	12.24	10.49	10.95
41-70 Inspections	16.42	23.89	14.94	15.44
71-100 Inspections	19.59	27.99	19.71	15.72



(a) *Score* (b) *Time Taken* (c) *Score/Runtime Fraction*

Figure 5.5: Comparison of Algorithms on Number of Inspections metric

datasets, whereas *PIRP-ILS* performs calculations faster in smaller datasets than *PIRP-C*, and comparatively struggles with larger datasets. This occurs because *PIRP-ILS* operates with a higher time complexity of n^2 than *PIRP-C* at $n * log_n$. No differing scoring trend was uncovered.

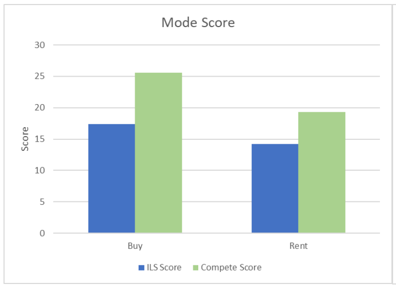
5.7 Inspection Mode

Another feature of the datasets was the mode of occupancy. Whether the inspection was for a rental or to buy:

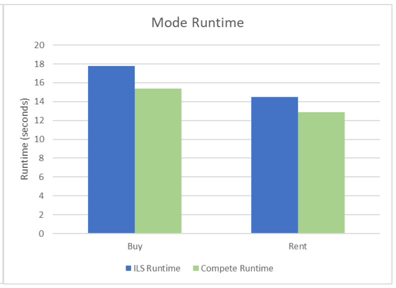
Table 5.6: Inspection mode results

Overall	PIRP-ILS Score	PIRP-C Score	PIRP-ILS Runtime	PIRP-C Runtime
Buy	17.39	25.58	17.77	15.40
Rent	14.25	19.32	14.52	12.86

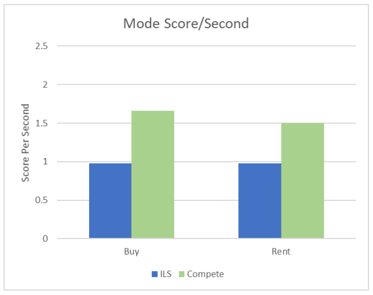
The inspection mode does not show any different trend in score or time taken than the overall case. *PIRP-C* exhibits superior performance for this data feature.



(a) Score



(b) Runtime



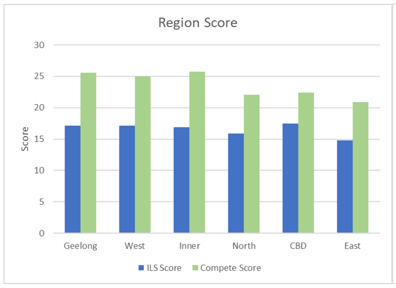
(c) Score/Runtime Fraction

Figure 5.6: Comparison of Algorithms on Mode metric

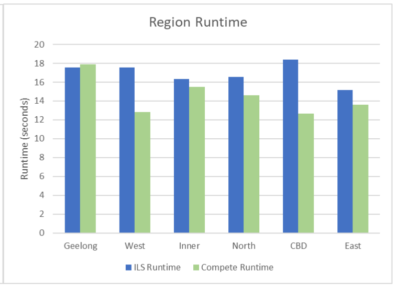
5.8 Region

Table 5.7: Region results

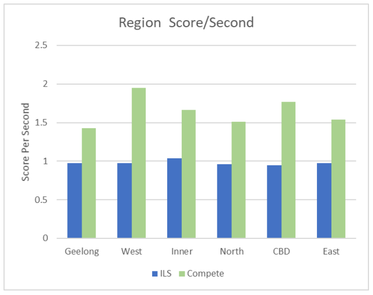
Overall	PIRP-ILS Score	PIRP-C Score	PIRP-ILS Runtime	PIRP-C Runtime
Geelong	17.12	25.54	17.55	17.86
West	17.12	24.97	17.56	12.81
Inner	16.90	25.75	16.33	15.49
North	15.89	22.09	16.55	14.59
CBD	17.49	22.37	18.41	12.64
East	14.76	20.90	15.16	13.60



(a) Score



(b) Runtime



(c) Score/Runtime Fraction

Figure 5.7: Comparison of Algorithms on Region metric

Similarly to inspection mode, the region does not show any different trend in score or time taken than the overall case. The trend of superior performance of the *PIRP-C* algorithm is bourne out by this data feature comparison.

5.9 Secondary API Analysis

The inclusion of a directions API is novel for travel time calculation in routing algorithms, we therefore consider the impact of its inclusion. First in terms of cost, second in terms of the accuracy of the travel time (and subsequent impact on each algorithm) and third on the overall runtime of each algorithm.

5.9.1 Costs

An analysis was conducted on the number of API calls and therefore the effective cost of a given plan with each of the algorithms. Based on a sample of 37 inspection dataset outputs, from both the *PIRP-C* and *PIRP-ILS* algorithms and a per call cost of US$0.005.

Table 5.8: Cost per algorithm run

Algorithm	Calls per run	Cost
PIRP-C	12	US$0.06
PIRP-ILS	34	US$0.17

The cost per run is 2.78 times higher in the *PIRP-ILS* algorithm than the *PIRP-C* algorithm. While both may bear an unacceptably high cost, the *PIRP-C* algorithm has considerably lower cost. Therefore, the effort to reduce this cost further in future work is easier with *PIRP-C*.

5.9.2 API impact

To see the effect that using an API has on each algorithm, we ran a sample of 60 datasets that were run in the original experiment. Both algorithms were altered to only call the euclidean travel time. The comparison results with the original experiment are below:

Table 5.9: Differences between Naive and API time estimation

Algorithm	Sensitivity	Mean Travel Time Difference (mins)
PIRP-C	Square Root	5.90
PIRP-C	Unchanged	3.41
PIRP-C	Squared	3.36
PIRP-ILS	Square Root	5.53
PIRP-ILS	Unchanged	3.97
PIRP-ILS	Squared	5.72

These travel time inaccuracies drive each algorithm to select different inspections, given the travel time is a significant factor in the selection of an inspection. In Table 5.8 we can see the score difference between both algorithms operating with the directions API vs. without the directions API.

The impact of the directions API is immense, for the more deterministic *PIRP-C* algorithm, without API, the algorithm runs on average for 14 milliseconds, whereas with the directions API will run up until the TimeLimit of 10 seconds, unless an error has occurred (e.g. no valid inspections), this results in an average runtime of 9.49 seconds.

Table 5.10: Differences between Niave and API

Algorithm	Mode	Mean Score	Mean Runtime (sec)
PIRP-C	With API	18.11	9.49
PIRP-C	Without API	19.52	0.01
PIRP-ILS	With API	11.47	26.38
PIRP-ILS	Without API	12.02	9.50

The short runtime of *PIRP-C* occurs because 30% of the unscheduled properties are assessed, once each, by the InsertC and NeighbourhoodReplace steps (the *PIRP-C Improve* phase. Whereas the *Improve* phase of the *PIRP-ILS* algorithm continues to execute until TimeLimit. The difference in calculation runtime is roughly 1,000 times between the *euclidean-only PIRP-C* calculation and *PIRP* algorithms with directions API calculation.

It is difficult to draw conclusions on why exactly the *PIRP-ILS* algorithm under-performs the *PIRP-C* in scoring. We assume that the narrower time windows leads to less directly competing inspections, which confound several *PIRP-ILS* operations such as RouletteWheel and Perturbation. These operations leverage stochasticity to escape local maxima. Further work will attempt to more broadly prove that stochastic solutions struggle to address highly time constrained routing problems.

5.10 Summary

We find that across all data features, *PIRP-C* performs better than *PIRP-ILS* regarding runtime and score. However, the feature algorithm stage in Section 5.3 leaves open the possibility of replacing the *Stage 2 - Improve* step of *PIRP-C* in future work. Table 5.11 summarises the overall results and features of each algorithm, *ILS*, *PIRP-ILS* and *PIRP-C*. Note, all the algorithm outputs, visualisation results, and summarisation files are available on public GitHub repository: https://github.com/ChrisBond-s3767838/PIRP-Minor-Thesis, available for reproducability. Note, that no private information about gathered inspections is available in any of the files present in the repository.

Table 5.11: Summary of Algorithm Differences between ILS, PIRP-ILS *and* PIRP-C

Category	ILS	PIRP-ILS	PIRP-C
Works for PIRP	✗	✓	✓
Faster runtime	N/A	✗	✓
Better score	N/A	✗	✓
API for travel time	✗	✓	✓
Alternative inspections presented	✗	✓	✓
Area Inspection	✗	✓	✓

Chapter 6

Conclusions

In this chapter we summarise the core contributions of our work. Planning out inspections to buy or rent a property, is a challenging and time consuming process, especially in a large city like Melbourne or Munich. An automated solution might be very useful, but the specifics of the problem domain makes it impossible to apply existing algorithms directly. Thus, either a novel algorithm, or domain-specific modification of some exiting algorithm is required.

In this Minor Thesis, we formalised this process as a domain called *Property Inspection Route Planning (PIRP)*. We specified it as a child-domain of an existing route planning domain, namely the *Orienteering Problem with Time Windows (OPTW)*, with a few important differences. In the *OPTW* domain, one of the commonly used algorithms is the *ILS* algorithm proposed by Gunawan et al. [22]. We modified it to be applicable to the *PIRP* problem, and named this modified version *PIRP-ILS*. This version represents the base-case for our analysis. We also proposed a novel approach focused on peer-competition named *PIRP-C*. This algorithm resolves to an ideal solution faster than the *PIRP-ILS* algorithm because restrictive time windows confound the stochasticity in *PIRP-ILS*. This faster resolution, leads to quicker runtimes and higher scores. All three algorithms *ILS, PIRP-ILS and* PIRP-*C* run using a two stage approach, with *Stage 1* as a initial solution construction stage, and *Stage 2* as a solution tuning stage.

The results of the experiment we conducted, indicate that across all data features *PIRP-C* operates with faster runtime and higher scores. Specifically, the *PIRP-ILS* algorithm seems to particularly struggle to perform as well as *PIRP-C*, on runtime and score results in datasets with between 71-100 inspections. This is likely due to the higher time complexity of *PIRP-ILS* at n^2, than *PIRP-C* at $n * log_n$. The lower score indicates that *PIRP-ILS* requires more runtime to resolve the problem adequately. However, in our experiment due to prohibitive API running costs, we reduced the runtime of both algorithms by reducing a parameter called TimeLimit from 60 seconds in *ILS* to 10 seconds. The TimeLimit value is checked after each improvement round in *Stage 2*. The results further indicate that the *Stage 1* of *PIRP-C* provides more score impact for time taken than *Stage 2*. *Stage 1* has the largest impact, we therefore propose future work to determine whether a further refinement of *Stage 2 of PIRP-C* might be possible.

Finally, we examined the impact of using a directions API. The results of our analysis demonstrated that there is enough accuracy gain to change the properties selected in both *PIRP-ILS* and *PIRP-C*, however that the added runtime of a direction API most effects the *PIRP-ILS* algorithm approach. Thus, the improved accuracy of using a directions API is essential to meet user expectations. Therefore, the novel *PIRP-C* approach is able to more closely meet user needs, as it leverages a direction API, and its score and runtime are optimised as compared with the base case *PIRP-ILS* algorithm. Note that a summary of all proposed future work is described in Appendix B.

To sum up, the novel *PIRP-C* algorithm is best suited to address the new domain *Property Inspection Route Planning (PIRP)*, that we introduce in our work.

Appendix A

Detailed Parameters

Geographic Location of property of interest

Data Format GeoJSON

Features Industry standard approach (formalised in Internet Engineering Task Force publication RFC 7946) to Geo-spatial Information System (GIS) information, see [8]. Used in popular GIS software (such as Google Maps, ArcGIS, QGIS Spotzi and Tableau) for addresses/locations (points), streets/boundaries (line strings), states/tracts of land (polygons), and multi-part collections of these. This standard offers flexibility of implementation, however we will limit our use solely to the points format, with only 3 elements, including: a name (which can be used for the text address), latitude and longitude.

The precise format is shown below:

```
var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "29/50 Rosslyn Street, West Melbourne",
        "area_inspection_duration":10:00,
        "property_inspection_duration_variance":1:20
    },
    "geometry": {
        "type": "Point",
        "coordinates": [-37.80725, 144.95257]
    }
};
```

Longitude and Latitude will be in float format, which will be used in fast pair-wise calculation of point distance, which will allow for categorisation of pairs into estimated travel time categories, where far point pairs will be excluded. *Address* will be in a text format allowing use in labelling of the property and pairwise navigation using map APIs.

Evaluation This approach is standardised, allowing interoperability with geo-spatial API services (such as Google Directions API). Encoding coordinates in this manner allows for a simplistic preliminary assessment of distance to classify whether a geographic pair should be explored, reducing API calls (which could reduce performance of the algorithm). Limiting the parameters to 3 fields simplifies the algorithm, which is essential for implementation in a new field.

Geographic Location of home location

Data Format GeoJSON (as above)

Evaluation We again use the same approach for a start-finish location, referred to here as home location. We imagine a typical user will depart from and return to their home once the plan has concluded.

Hedonic preference rating of inspection

Data Type Integer

Description Each inspection is given a hedonic rating. This rating is used to determine the recommended and competing inspections in the plan.

Evaluation A hedonic rating (e.g. 1-4) was chosen in favour of an absolute ranking scheme (e.g. 1-100) as it offered a simpler approach to users and is more in line with industry practice than absolute rankings, see [28]. Drawbacks of this approach include a loss of granularity, as users cannot define preference between similarly scored inspections, and more 'tie-breaking' occurrences (where the cumulative value of two competing routes are exactly equivalent). Conversely there is a reduced cognitive load on the user (as they compare against properties they may not remember well), a more challenging approach to weighting each properties importance, difficulty in determining preference grouping and some users investing significant time into the ranking process. Both approaches require less detail from the user than the "Detailed" approach in Figure A.1.

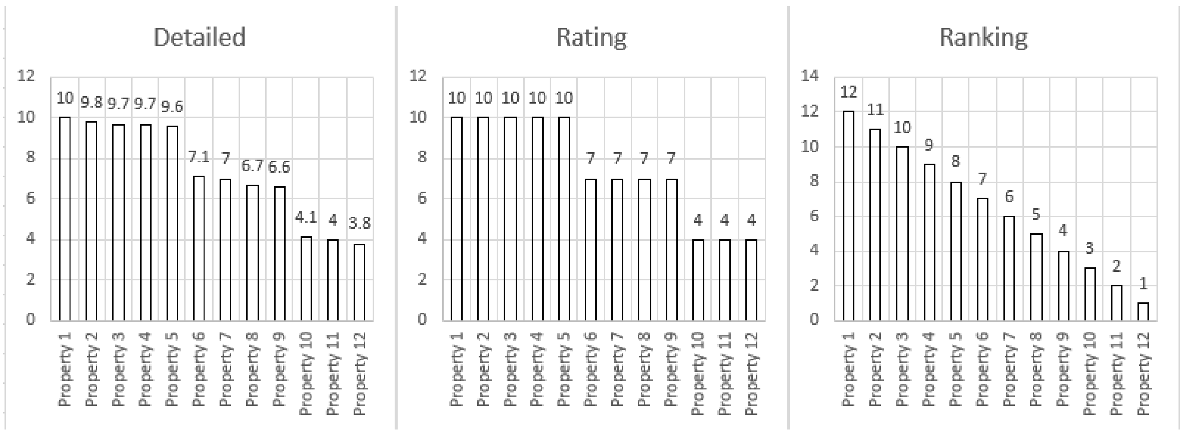


Figure A.1: User Alteration to Manual Solution

Inspection Start Time

Data Type DateTime

Description Provides a start time for inspections, this represents a period before which an inspection cannot commence. If a person arrives at the location before the start time they will have to wait.

Evaluation The inclusion of a start time ensures inspections do not start before their listed start time, therefore

Inspection Finish Time

Data Type DateTime

Description Provides a finish time for inspections, represents a period after which an inspection must cease. For an inspection to be valid, the inspector must attend the inspection location at a time that allows the standard inspection duration to occur prior to the inspection finish time.

Plan start time

Data Type DateTime

Description Provides a start time for the overall plan, this represents a period before which, no inspections can commence. All inspections that fall outside this time are removed from the unscheduled inspection list.

Evaluation The inclusion of a plan start time, limits the plan to occur within more realistic timeframes for the user.

Inspection Finish Time

Data Type DateTime

Description Provides a finish time for the overall plan, this represents a period after which, no inspections can commence. All inspections that fall outside this time are removed from the unscheduled inspection list.

Evaluation The inclusion of a plan finish time, limits the plan to occur within more realistic timeframes for the user.

The inspection day selected

Data Type Date

Description Provides a date for inspections, allowing for filtering of the inspections that occur on a specific date.

Evaluation By limiting inspection routing to a single day, this reduces the complexity of the algorithm and enables better wall-time performance for an algorithm that is focussed on applying

The standard inspection duration

Data Type Integer (seconds)

Description Provides a duration that the inspections will take. Once the inspector has made it to the inspection location, the duration must elapse prior to the inspection finish time for it to be a valid inspection. The inspection time is represented by a seconds integer.

Evaluation An inspection duration is required to provide a viable time window for inspection. The time window requires the inspection to start before latest start time which is defined as [inspection finish time] - [standard inspection duration]. Requiring a general user parameter rather than defining a property specific parameter n times, only requires definition once. Unfortunately this approach limits the time spent

The travel mode

Data Type enum

Description The main mode of transport between inspections. The selection is between public transport and driving. This is used to determine the routing approach.

Evaluation A requirement for an approach that can operate in high-density locations where public transport is more realistic than car use for travelling to inspections. Further, some users may not have a private vehicle, or prefer the public transport option.

Selection of a default walking time for sub-10 minute travel

Data Type Boolean

Description The user has the option to nominate to default to walking where the travel time via walking is below 10 minutes. A True value here indicates that walking will be the travel mode for travel where the start and finish locations are within 10 minutes walking. A False value indicates that the preferred travel mode will remain the chosen mode for travel where the start and finish locations are within 10 minutes walking.

Evaluation Switching to walking supports more sparing use of the primary transport mode, which may not make practical sense with parking and public transport delays being a common feature in many Australian cities. However a shortcoming of this approach is that different users would have different preferences regarding the time-threshold for switching to walking. A standard threshold was used to simplify the user interface.

Travel Time

Data Type Double

Description The final travel time between the previous inspection, and the current inspection will be determined using a directions API.

Evaluation In Section 5.9.2, we show that euclidean travel times can produce average travel time difference of up to 5.9 minutes as compared with a directions API. It also shows that this results in different inspections being added to the plan. Therefore, to achieve a maximally accurate plan, a directions API is used, and the time unit is minutes, represented by *double* format. This allows for relatively detailed time calculations to be made.

Area inspection default duration

Data Type Integer

Description The area duration added to the inspection (if not previously satisfied) is a standard integer value of either 10 minutes for inner suburban and CBD or 20 minutes for regional or outer suburban regions.

Evaluation This provides a simple, consistent approach to area inspections in each region. However, detailed travel time to a chosen amenity, and potential user adjustment is defined as future work in Appendix B.

Output

Based on the property of interest, the property location and inspection window details will be retrieved from the online listing service. The user will receive:

- A selection of inspection to attend
- A list of alternative inspections than can be substituted for primary inspections

Below in Figure A.2 is an example of user adjustment, based on the information in Figure 1.2. Here the user has clicked on *29/50 Rosslyn Street, West Melbourne* (row 3) instead of the algorithm selected *29 Watt Street, Spotswood* (row 5) [selected in Figure 1.2]. This means that James can't make it to *5/29 Dover Road, Williamstown* (row 9) because of the longer travel time, but now he can make it to *10/171 Flemington Road, North Melbourne* (row 7) because of reduced travel time. User adjustment is essential because users do not display consistency between what they project they will prefer and their actual preferences. Further, the standard weighting methodology applied to property rankings will imperfectly reflect user's preference. The approach described here maximises user choice, enabling users to adjust their plan, it also allows them to appreciate trade-offs in real time, without being overwhelmed by combinatorial complexity. The inspection process is already overwhelming, users like James would benefit from reduced complexity, minimising the symptoms of effortful decision making.

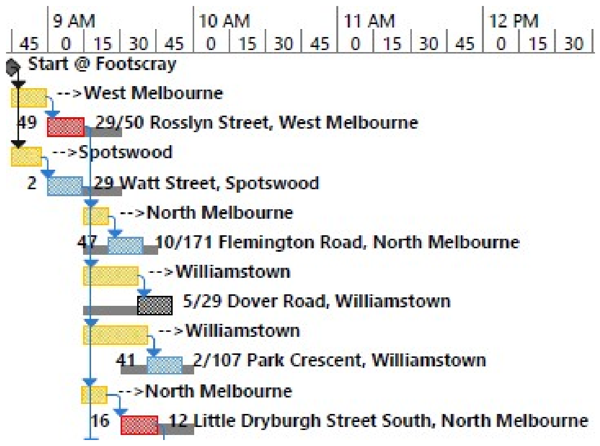


Figure A.2: User Alteration to Manual Solution

Appendix B

Future Work

This appendix summarises the future work expressed throughout this paper, the key ones are:

- multi-day plans
- integrate with recommendation services
- live plan navigation
- allow higher/unconstrained TimeLimit for comparison
- user editable inspection durations
- user defined amenity
- reduce cost of directions API
- different geographic areas
- alternative Stage 2 operation for *PIRP-C*

Multi-day plans for simplicity, the *PIRP* problem has been limited to single day plans, in future work we propose multi-day plans. This is valuable because individuals may select properties which have inspections on different days. Further, multiple day plans can avoid repetition in already inspected properties.

Integrate recommendation engine into the selection of properties. It is a labourious process to mark every property with a score, recommendation engines are a proven approach to reduce this manual user burden.

Higher TimeLimit for cost reasons the TimeLimit was reduced from 60 seconds to 10 seconds. It is likely that this has a negative impact on the *PIRP-ILS* score, by testing how *PIRP-ILS* behaves at different TimeLimit values, we can appreciate when *PIRP-ILS* resolves to its optimal score, and better analyse the strengths and drawbacks of this algorithm.

Editable durations in this paper we have constrained inspection durations to three values, 10, 15 and 20 minutes. This simulates different user categories, in reality users will want to set their own average inspection time.

User defined amenity here we have selected the large grocery store amenity for each location, given its relatively ubiquitous appeal. However, different users will have different amenities that they prefer. These were not incorporated into the *PIRP* domain because it was a manual, and time consuming process to define the grocery store. Further, in the *PIRP* solutions offered in this paper, a default area inspection time is provided, additional future work is required to record specific travel time to the chosen amenity for each inspection. Defining additional amenities will require a novel approach to limit the effort of defining appropriate locations and accurately interpret user amenity requirements.

Reduce directions API cost the cost of \$0.06 for *PIRP-C* and \$0.17 for *PIRP-ILS* for each run may be prohibitive in a product offering setting. Therefore, approaches to get more value out of the directions.

Different geographic areas Two thirds of Australia's population lives in capital cites, and 83% of the population living in the top 50 cities in Australia, see [42]. Therefore the inclusion of Melbourne (the Capital City of Victoria) [39] and Geelong the largest Victorian regional city and 14th largest city in Australia [19], provides a decent approximation of the majority of the Australian market for housing. However, there are likely differences to each location, so future work should encompass smaller regional and rural settings and other major cities, such as Sydney.

Alternative Stage 2 operation for PIRP-C *PIRP-C* operates a high-scoring and short runtime *Stage 1* construct operation, its *Stage 2* has a relatively lower score impact for its cost and runtime, as seen in Section 5.3. A focus on an alternate *Stage 2* operation could reduce the time and cost of this operation, or potentially increase the score provided, or both.

Appendix C

Area Inspection Demarcation Approach

Selecting the relevant amenity for each inspection followed several rules below:

- Amenity type
- Maximum travel time
- Travel mode
- Selection method from competitors

Amenity type we selected grocery stores as the amenity of choice. Given different demographics prefer different amenity types, it was challenging to find one that linked all demographics. An example of some of the potential amenities are pubs, healthcare, schooling, highways, jobs precinct, public transit, restaurants, local walkability, cafes, cycling paths, parks or nature reserves. However with 91% of people buying groceries more than once a month, see [53], this is the most ubiquitous of the above amenities. In future work, we will allow users to define the amenity most important to them.

Maximum travel time given we see increased density of properties closer to the CBD, we can assume that individuals inspecting in inner locations would expect a quicker travel time to their chosen amenity. We found in a sample of our dataset, that properties averaged 1.2 km away from each other in the CBD, 5.1 km in inner-suburban, 11.7 km in outer suburban and 10.4 km in regional areas. Therefore, for the CBD and inner-sububan regions, the maximum travel time to a local amenity was 10 minutes. For outer-suburban and regional locales the maximum travel time allowed was 20 minutes.

Travel mode as covered in Section 3.2, the travel mode in the CBD is public transport, whereas other locations use car as the primary travel mode. Therefore, for logical consistency we echo this approach for travel to local amenity as an individual would not abandon their car to take public transport to their local amenity, and vice versa if already taking public transport, would not have their car available to take to their local amenity.

Selection method from competitors for many inspections, several grocery stores likely fall inside the travel radius defined above. To compare, firstly supermarkets are preferred to convenience and small grocers. If this does not split the competitors, then the nearest store is chosen.

Bibliography

[1] Rahim A Abbaspour and Farhad Samadzadegan. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications*, 38(10):12439–12452, 2011.

[2] Aldair Alvarez, Pedro Munari, and Reinaldo Morabito. Iterated local search and simulated annealing algorithms for the inventory routing problem. *International Transactions in Operational Research*, 25(6):1785–1809, 2018.

[3] Pablo Alvarez, Iosu Lerga, Adrian Serrano-Hernandez, and Javier Faulin. The impact of traffic congestion when optimising delivery routes in real time. a case study in spain. *International Journal of Logistics Research and Applications*, 21(5):529–541, 2018.

[4] Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Computers & Operations Research*, 107:32–42, 2019.

[5] Australian Bureau of of Statistics. Media Release - Census: younger Australians more likely to make a move (Media Releases), 2017, Oct 2017.

[6] Dominic Beattie. Housing statistics in australia: Home ownership & rent. `https://www.savings.com.au/home-loans/by-the-numbers-australian-home-ownership-tenancy-statistics#homeownership`, Aug 2019. Accessed: 15 Oct 2021.

[7] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

[8] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub, and T. Schaub. The GeoJSON Format. RFC 7946, August 2016.

[9] Vicente Campos, Rafael Martí, Jesús Sánchez-Oro, and Abraham Duarte. Grasp with path relinking for the orienteering problem. *Journal of the Operational Research Society*, 65(12):1800–1813, 2014.

[10] I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996.

[11] Alber J Christianto, Peng Chen, Osheen Walawedura, Annie Vuong, Jun Feng, Dong Wang, Maria Spichkova, and Milan Simic. Enhancing the user experience with vertical transportation solutions. *Procedia computer science*, 126:2075–2084, 2018.

[12] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks: An International Journal*, 30(2):105–119, 1997.

[13] Wu Deng, Junjie Xu, and Huimin Zhao. An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. *IEEE access*, 7:20281–20292, 2019.

[14] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.

[15] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation science*, 39(2):188–205, 2005.

[16] Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, and Grammati Pantziou. Mobile recommender systems in tourism. *Journal of network and computer applications*, 39:319–333, 2014.

[17] Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, and Grammati Pantziou. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, 20(3):291–328, 2014.

[18] Maha Gmira, Michel Gendreau, Andrea Lodi, and Jean-Yves Potvin. Tabu search for the time-dependent vehicle routing problem with time windows on a road network. *European Journal of Operational Research*, 288(1):129–140, 2021.

[19] Australian Commonwealth Government. Population projections - city of greater geelong. `https://data.gov.au/data/dataset/geelong-population-projections`, Jul 2015. Accessed: 10 Jan 2022.

[20] Aldy Gunawan, Hoong Chuin Lau, and Kun Lu. An iterated local search algorithm for solving the orienteering problem with time windows. In *European conference on evolutionary computation in combinatorial optimization*, pages 61–73. Springer, 2015.

[21] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[22] Aldy Gunawan, Hoong Chuin Lau, Pieter Vansteenwegen, and Kun Lu. Well-tuned algorithms for the team orienteering problem with time windows. *Journal of the Operational Research Society*, 68(8):861–876, 2017.

[23] Melissa Heagney. Melbourne's most and least walkable suburbs. `https://www.domain.com.au/healthy-melbourne/melbournes-healthy-suburbs-2018/melbourne-most-least-walkable-759248/`, Oct 2018. Accessed: 15 Dec 2021.

[24] Eduard Hromada et al. Analysis of relationship between market value of property and its distance from center of capital. *Czech Technical University: Prague, Czech Republic*, 2018.

[25] Liangjun Ke, Laipeng Zhai, Jing Li, and Felix TS Chan. Pareto mimic algorithm: An approach to the team orienteering problem. *Omega*, 61:155–166, 2016.

[26] SW Kramer and JL Jenkins. Understanding the basics of cpm calculations: What is scheduling software really telling you. In *PMI® Global Congress*, 2006.

[27] Jin Li. Research on team orienteering problem with dynamic travel times. *JSW*, 7(2):249–255, 2012.

[28] Juyun Lim. Hedonic scaling: A review of methods and theory. *Food quality and preference*, 22(8):733–747, 2011.

[29] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. Personalized trip recommendation for tourists based on user interests, points of interest visit durations and visit recency. *Knowledge and Information Systems*, 54(2):375–406, 2018.

[30] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[31] Achmad Mustofa Luthfi, Nyoman Karna, and Ratna Mayasari. Google maps api implementation on iot platform for tracking an object using gps. In *2019 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pages 126–131. IEEE, 2019.

[32] Shawn Mathew and Joshy Joseph. Decision fatigue. *Listening to Consumers of Emerging Markets*, page 175, 2014.

[33] Ines Mathlouthi, Michel Gendreau, and Jean-Yves Potvin. Branch-and-price for a multi-attribute technician routing and scheduling problem. In *SN Operations Research Forum*, volume 2, pages 1–35. Springer, 2021.

[34] Roy Morgan. Realestate.com.au still leads with 4.5m visitors vs 3.2m going to domain – but 2.2m australians use both. `http://www.roymorgan.com/findings/6881-digital-property-search-audiences-for-realestate-vs-domain-australia-march-2016-201`, Jul 2016. Accessed: 18 Dec 2021.

[35] Simone Mueller, IL Francis, and Larry Lockshin. Comparison of best–worst and hedonic scaling for the measurement of consumer wine preferences. *Australian Journal of Grape and Wine Research*, 15(3):205–215, 2009.

[36] Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[37] Jakob Nielsen. `https://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/`, Aug 2001. Accessed: 01 Sep 2021.

[38] Justin O'Beirne. How many people use google maps compared to apple maps? `https://www.justinobeirne.com/how-many-people-use-google-maps-compared-to-apple-maps`, Jul 2021. Accessed: 12 Sep 2021.

[39] Bureau of Statistics. Regional population, 2020-21 financial year. `https://www.abs.gov.au/statistics/people/population/regional-population/latest-release`, Mar 2022. Accessed: 11 Jan 2022.

[40] Claudia Pegler, Hankan Li, and Dorina Pojani. Gentrification in australia's largest cities: A bird's-eye view. *Australian planner*, 56(3):191–205, 2020.

[41] Michael P Peterson. Evaluating mapping apis. In *Modern Trends in Cartography*, pages 183–197. Springer, 2015.

[42] Centre for Population. `https://population.gov.au/population-topics/topic-population`, Jan 2022. Accessed: 15 Feb 2022.

[43] Giovanni Righini and Matteo Salani. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & operations research*, 36(4):1191–1203, 2009.

[44] Zülal Sevkli and F Erdogan Sevilgen. Discrete particle swarm optimization for the orienteering problem. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[45] Yang Shen, Mingde Liu, Jian Yang, Yuhui Shi, and Martin Middendorf. A hybrid swarm intelligence algorithm for vehicle routing problem with time windows. *Ieee Access*, 8:93882–93893, 2020.

[46] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[47] Australian Bureau of Statistics. Data by region, Jun 2020. Accessed: 16 Oct 2021.

[48] Theodore Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[49] Fahim Ullah, Samad Sepasgozar, and Siddra Siddiqui. An investigation of real estate technology utilization in technologically advanced marketplace. In *9th International Civil Engineering Congress (ICEC-2017), "Striving Towards Resilient Built Environment", December*, pages 22–23, 2017.

[50] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009.

[51] Cédric Verbeeck, Kenneth Sörensen, E-H Aghezzaf, and Pieter Vansteenwegen. A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research*, 236(2):419–432, 2014.

[52] Cédric Verbeeck, Pieter Vansteenwegen, and El-Houssaine Aghezzaf. The time-dependent orienteering problem with time windows: a fast ant colony system. *Annals of Operations Research*, 254(1):481–505, 2017.

[53] Sophie Wallis. Supermarket statistics 2021: Finder. `https://www.finder.com.au/supermarket-statistics-2021`, Nov 2021. Accessed: 18 Dec 2021.

[54] Fengrui Yu, Xueliang Fu, Honghui Li, and Gaifang Dong. Improved roulette wheel selection-based genetic algorithm for tsp. In *2016 International conference on network and information systems for computers (ICNISC)*, pages 151–154. IEEE, 2016.

[55] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. Time-aware point-of-interest recommendation. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 363–372, 2013.

[56] Lei Zhu and Jeffrey D Gonder. A driving cycle detection approach using map service api. *Transportation Research Part C: Emerging Technologies*, 92:349–363, 2018.