University of Basel

Evaluating the Cyclic Landmark Heuristic with a Logistics-specific Landmark Generator

Bachelor thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artifical Intelligence Research Group
https://ai.dmi.unibas.ch/

Examiner: Prof. Dr. Malte Helmert
Supervisor: Clemens Büchner

Günes Aydin
guenes.aydin@unibas.ch
2018-058-636

September 12, 2021

Acknowledgments

I want to thank my supervisor Clemens Büchner for assisting me in my thesis, responding to my questions, and always trying to help for any concerns. As his first student, he was exceptionally helpful with spot-on feedback, which I appreciate greatly. Additionally, I want to thank Prof. Dr. Malte Helmert for allowing me to write a bachelor thesis in his research group artificial intelligence and also allowing me to contribute to something valuable. Furthermore, I want to give a big thanks to Kevin Waldock for his general input on implementation and design. I also want to thank my friends and family for supporting me during these times and I am thankful for any advice and help.

Abstract

In this thesis, we generate *landmarks* for a *logistics-specific* task. Landmarks are actions that need to occur at least once in every plan. A landmark graph denotes a structure with landmarks and their edges called *orderings*. If there are *cycles* in a landmark graph, one of those landmarks needs to be achieved at least twice for every *cycle*. The generation of the logistics-specific landmarks and their orderings calculate the *cyclic landmark* heuristic. The task is to pick up on related work, the evaluation of the cyclic landmark heuristic. We compare the generation of landmark graphs from a domain-independent landmark generator to a domain-specific landmark generator, the latter being the focus. We aim to bridge the gap between domain-specific and domain-independent landmark generators. In this thesis, we compare one domain-specific approach for the logistics domain with results from a domain-independent landmark generator. We devise a unit to pre-process data for other domain-specific tasks as well. We will show that specificity is better suited than independence.

Table of Contents

Acknowledgments **ii**

Abstract **iii**

1 Introduction **1**

2 Background **3**

2.1 Definitions 3

2.2 Heuristic Search 4

3 Logistics Domain **5**

3.1 Logistics Tasks 5

4 Landmarks **8**

4.1 Landmarks and Ordering 8

5 Cyclic Landmark Heuristic **12**

6 Landmark Generator **14**

6.1 Generating landmarks 14

7 Experimental Evaluation **18**

7.1 Initial heuristic value 18

7.1.1 Logistics00 and additional benchmark 19

7.1.2 Logistics98 20

7.2 Total Time 22

7.2.1 Logistics00 and additional benchmark 22

7.2.2 Logistics98 23

7.3 Expansions until last jump 25

7.3.1 Logistics00 and additional benchmark 25

7.3.2 Logistics98 25

7.4 Memory 27

7.4.1 Logistics00 and additional benchmark 27

7.4.2 Logistics98 28

7.5 Comparison to Paul et al. (2017) 29

8 Conclusion **30**
8.1 Future Work 30

Appendix A Appendix **31**
A.1 Benchmarks and their structure 31
A.2 Information Assembling 31

Bibliography **34**

Declaration on Scientific Integrity **36**

1 Introduction

How exciting! You just ordered something from the internet and your parcel/package is on the way home to you. The post office identifies a plan to ship your package in no time. Thinking about an optimal plan to efficiently deliver a package can be hard.
Classical planning is an approach to find optimal or satisfying solutions to a given problem. Given a state of a specific problem, what is the most optimal way by applying different actions to the goal, your home? A sequence of actions will be a *plan* for this specific state. A cost-efficient plan or an optimal cost plan minimizes the cost to solve a problem. A problem is for example the logistics task: your parcel that needs to be shipped to your home. We use search algorithms to find the optimal path from the start location to the goal state, your home. Search algorithms use heuristics as a guidance tool, a function that maps a state to a real non-negative number and traverses a state space by applying several actions until a goal is found or it can be shown that no such plan exists.
It is clear, that your home is the goal. You want your package at some point in time. This idea is conceptualized as a *landmark*: an action that needs to happen in every planning task plan. The delivery man has to pick up your package at the packing center before he can deliver it to you. This dependency is called a *landmark ordering*. They provide a sequence, which action has to be executed at which point.
Let's consider the following scenario: You also want to ship a package back to the provider and you await a package your own at the same time. The delivery man picks up your parcel at the packing center, delivers it to you, picks up your package, and brings it back to the packing center for further processing. Such an occurrence is called a *cycle*, which will lead to an improvement of the heuristic estimate. The cyclic heuristic is based on the findings of Paul and Helmert (2016) and elaborated further by Paul et al. (2017) as well as from Büchner et al. (2021a). With the use of domain-independent landmark generators, Büchner et al. (2021a) shared their results in these papers. They noticed that domain-independent landmarks yield worse results than domain-specific. The focus of this thesis is to generate landmarks and create a landmark graph, based on their checklists, and recreate the logistics-specific landmark generator for the domain-independent heuristic. We provide the theoretical basics and understanding of classical planning in chapter 2 and formalize the logistics task in chapter 3, as well as the basics of landmarks in chapter 4. The calculation

of the cyclic landmark heuristic is thematized in chapter 5, and the creation of the landmark graph is covered in chapter 6. Lastly, we show in an experiment the advantages of domain-specific landmark generators over domain-independent landmark generators and finish this thesis with a conclusion.

2 Background

Before starting with the main topic, we want to briefly explain some relevant definitions to get a conceptual background and understanding of what we will use during this thesis.

2.1 Definitions

We consider a SAS^+ planning task as defined by Richter (2010):

Definition 1 (Planning Task) A SAS^+ planning task is a 4-tuple $\langle V_{state}, s_0, s_\star, A\rangle$ with the following components:

- V_{state} is a finite set of state variables, each with an associated finite domain D_v. A fact is a pair $\langle v, d\rangle$ (also written $v \mapsto d$), where $v \in V_{state}$ and $d \in D_v$. A partial variable assignment s is a set of facts, each with a different variable. A state is a variable assignment defined on all variables V_{state}.
- s_0 is a state called the initial state.
- $s_\star$ is a partial variable assignment called the goal.
- A is a finite set of actions (also referred to as operators), each associated with two partial variable assignments *pre(a)* and *eff(a)*. The facts in *pre(a)* and *eff(a)* are called the preconditions and effects of action $a \in A$, respectively. Each action furthermore has an associated non-negative cost $cost(a)$.

An action is *applicable* in a state s if *pre(a)* $\subseteq s$. Applying an applicable action a will result into state $s' = s[a]$, where *eff(a)* $\subseteq s'$. A sequence of actions π is applicable if all actions a_i are applicable in s_i for $i = 1, \ldots, n$. If an action sequence includes an action which leads to a goal state and $s_* \subseteq s_o[\pi]$, it is considered as a *plan*. The cost of π is the sum over all $costs(a_i)$. A plan is called optimal, if its costs is minimal among all plans. A *state space*, a full list of all states and their possible actions, is induced by the planning task.

Definition 2 (State spaces) A state space is induced by the planning task and it is a 6-tuple $\mathcal{S} = \langle S, A, cost, T, s_0, S^\star \rangle$ with:

- S : a finite set of states.
- A : a finite set of actions.
- $cost : A \to \mathbb{R}_0^+$, a cost function that maps an action to a non-negative real number.
- $T \subseteq S \times A \times S$ a transition relation.
- $s_0 \in S$ the initial state.
- $S^\star \subseteq S$ a set of goal states.

The cost function is considered optimal if the path costs from state s to $s^\star \in S^\star$ are minimal.

2.2 Heuristic Search

There are several possibilities to solve a planning task, but the most common one is the heuristic search.

Definition 3 (Heuristic Search) A heuristic function or heuristic is a function that maps a given state $s \in S$ to a real non-negative number or infinity:

$$h : S \to \mathbb{R}_0^+ \cup \{\infty\} \tag{2.1}$$

Following are some important properties a heuristic can have:

- A perfect heuristic is called $h^\star$ which maps all states to the optimal cost value, the minimal path costs.
- A heuristic is called *admissible*, if all states share this property: $h(s) \leq h^\star(s)$.
- A heuristic is called *safe*, if $h^\star = \infty$ for all states with $h(s) = \infty$.

Search algorithms use this heuristic function to evaluate the next promising actions. In this thesis, A* (Hart et al., 1968) will be the search algorithm of choice. The concept of A* is to expand nodes based on their path cost and heuristic value. Since we are searching for an optimal plan, it will always expand nodes with minimal cost until it either finds the first goal state, or runs into a dead-end (where $h(s) = \infty$) when expanding all possible paths. A* guarantees to find an optimal plan with an admissible heuristic.

3 Logistics Domain

Whilst all planning tasks are defined the same way, we differentiate between different classes of problems. Tasks from the same *domain* are similar to each other. In this thesis, we focus on the *logistics* domain (McDermott, 2000) and we will introduce it in this chapter.

3.1 Logistics Tasks

In a logistics task, we want to optimally transport packages from their origin to its destination. Therefore we have two transportation methods, trucks, and airplanes. Trucks are used to deliver packages between locations in a city, so-called intra-city transportation. Airplanes are inter-city transportation methods that can travel between airports. There may be multiple cities with different locations.
From Paul et al. (2017) we get the following definition:

Definition 4 (Logistics Task) A logistics task is given as a tuple $\langle L, C, P, A, city, airport, origin, dest \rangle$, where

- L *is a finite set of* locations,
- C *is a finite set of* cities,
- P *is a finite set of* packages,
- T *is a finite set of* trucks,
- A *is a finite set of* airplanes,
- $city : L \to C$ *assigns each location a city,*
- $airport : C \to L$ *assigns each city an airport location in this city, i.e. city(airport(c)) = c for all* $c \in C$,
- $origin : P \cup T \cup A \to L$ *specifies the origin location of each package, truck and airplane, where the origin of an airplane is always an airport location, and*
- $dest : P \to L$ *defines a destination for each package.*

A vehicle can either be a truck or airplane, therefore introducing a set *vehicles* $V_{vehicle} = T \cup A$.
For a further description in this thesis, we use the following convention, to avoid more formalities:

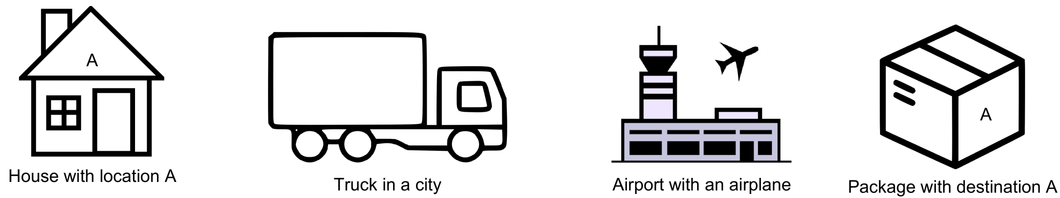

Figure 3.1: Description of the Logistics Task

Note that the airplane is not used in combination with the airport, it is a set on its own, as defined above. Airports do not require to have an airplane waiting there.
Both vehicles $v \in V_{vehicle}$ have the property to load or unload a package. To load a package the truck or the airplane needs to be at the same position as the package, e.g. one can only load a package into an airplane at an airport. The truck has the addition, that it can drive to locations within the same city. The airplane however can only fly in-between cities. In total four operators are applicable where three of them can be used by the truck and three of them by the airplane:

- *load(v,p,l)*, where $v \in V_{vehicle}$, $p \in P$ and $l \in L$, means loading a package p at position l in to vehicle v.
- *drive(t,l)*, where $t \in T$ and $l \in L$, driving the truck t from its current position to the desired location l.
- *unload(v,p,l)* where $v \in V_{vehicle}$, $p \in P$ and $l \in L$, means unloading a package p at position p from vehicle v.
- *fly(a,l)*, where $a \in A$ and $l \in L$, flying with the airplane from its current position to the desired location l.

Applying a sequence of these operators creates a plan for a logistics domain. An example illustration with an example plan is found below:

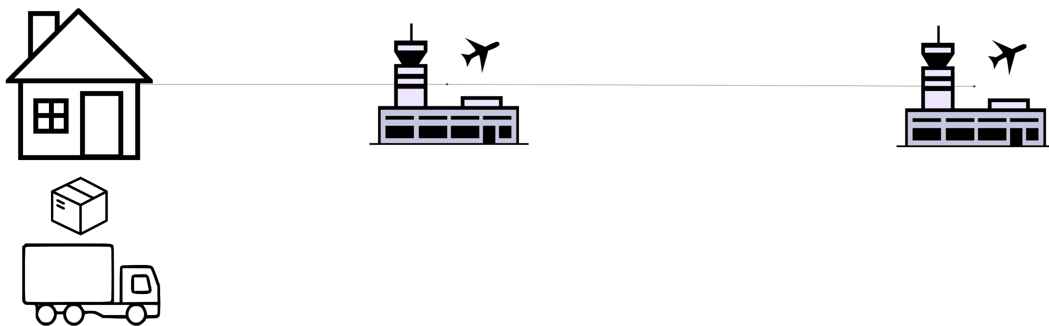

Figure 3.2: An example of a simple plan

1. *load(t,p,l_1)* the package p into the truck t at location l_1
2. *drive(t,l_2)* to the airport $l2$
3. *unload(t,p,l_2)* the package p at the airport l_2 from truck t
4. *load(a,p,l_2)* the package p in to the airplane a at location/airport l_2
5. *fly(a,l_3)* to the destination city
6. *unload(a,p,l_3)* the package p at the airport l_3 from airplane a

The planning task for Fig. 3.2.

Since we need to transport packages from their origin to their destination, we will need to use the given transportation tools. In Fig. 3.2, the truck needs to drive to the airport with the package and the airplane flies to the destination city to deliver the package. These edges are used to describe a delivery graph. We distinguish two different delivery graph, defined in Paul et al. (2017):

Definition 5.1 (Truck Delivery Graph)
For a given state s of a logistics task $\langle L, C, P, T, A, city, airport, origin, dest\rangle$ and $c \in C$, the truck delivery graph for c is the directed graph $D_s^c = (V_{graph}, E)$, where

- $V_{graph} = \{l \in L \mid city(l) = c\}$ are the locations in city c, and
- E contains the following edges for each package p with $pos_s(p) \neq dest(p)$:
  - If $city(pos_s(p)) = city(dest(p)) = c$, then there is an edge $pos_s(p) \rightarrow dest(p)$.
  - If $city(pos_s(p)) = c$, $city(dest(p)) \neq c$ and $pos_s(p) \neq airport(c)$ there is an edge $pos_s(p) \rightarrow airport(c)$.
  - If $city(pos_s(p)) \neq c$, $city(dest(p)) = c$ and $dest(p) \neq airport(c)$ there is an edge $airport(c) \rightarrow dest(p)$

Definition 5.2 (Airplane Delivery Graph) For the airplane delivery graph we have:
For state s of logistics $\langle L, C, P, T, A, city, airport, origin, dest\rangle$, the airplane delivery graph is the directed graph $D_s^A = (C, E)$ where $E = \{(c, c') \mid \text{there is a } p \in P\, s.t.c = city(pos_s(p)) \neq city(dest(p)) = c'$.

4
Landmarks

The main topic of this thesis is to generate landmarks for the cyclic landmark heuristic. Landmarks are originally introduced as facts that need to occur at least once in every planning task. For example, using the logistics domain, if a package needs to be delivered to point B however its position is still at A, a truck or an airplane, if it's intercity transportation, has to be at the position of the package at some point in time. In this chapter, we want to explain some fundamental definitions.

4.1 Landmarks and Ordering

As from Richter and Westphal (2010), we get the following definition for landmarks:

Definition 6 (Landmarks)
Let $\Pi = \langle V, s_0, s_\star, A \rangle$ be a planning task in finite-domain representation, let $\pi = \langle o_1, \ldots, o_n \rangle$ be an operator sequence applicable in s_0, and let $i, j \in \{0, \ldots, n\}$.

- A propositional formula φ over the facts of Π is called a **fact formula**.
- A fact F is true at time i in π iff $F \in s_0[\langle o_1, \ldots, o_n \rangle]$.
- A fact formula φ is true at time i in π iff φ holds given the truth value of all facts of Π at time i. At any time $i < 0$, φ is not considered true.
- A fact formula φ is a *landmark* of Π iff in each plan for Π, φ is true at some time.
- A propositional formula φ over the facts of Π is added at time i in π iff φ is true at time i in π. but not at time i-1 (it is considered added at time 0 if it is true in s_0).
- A fact formula φ is first added at time i in π iff φ is true at time i in π, but not at any time $j < i$.

An important thing to notice is, that everything in the initial state and the goal state are considered landmarks since they are always true at some time. For example, if the truck is at position B in the initial state or delivering the package at some time, are considered landmarks.

As of right now, we only talked about single trucks in cities or single airplanes between cities with illustration, not per definition. However, there is also the possibility that there is more than just one truck in cities or one airplane between cities. Therefore the term *disjunctive* fact landmarks is introduced. These are defined as sets of facts (note that a fact is a pair $\langle v, d \rangle$) where at least one holds at some time.

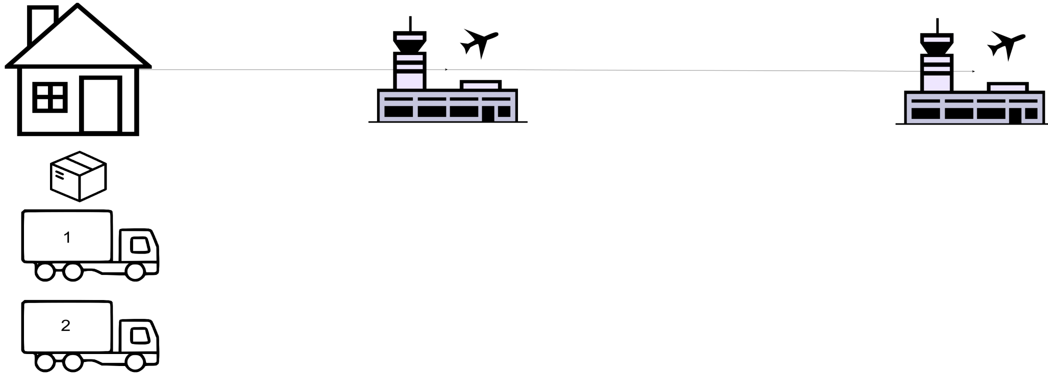

Figure 4.1: A small example with disjunctive fact landmarks. Package p can be delivered by truck 1 or truck 2. The disjunctive landmark however would be the set of those two facts combined.

In example 4.2 what can be seen is, if the $dest(p) \neq pos_s(p)$ and $s(t) = l \neq pos_s(p)$, the truck has to pick up the package before it can deliver it at some point. This can be conceptualized in a formal definition called *orderings*.

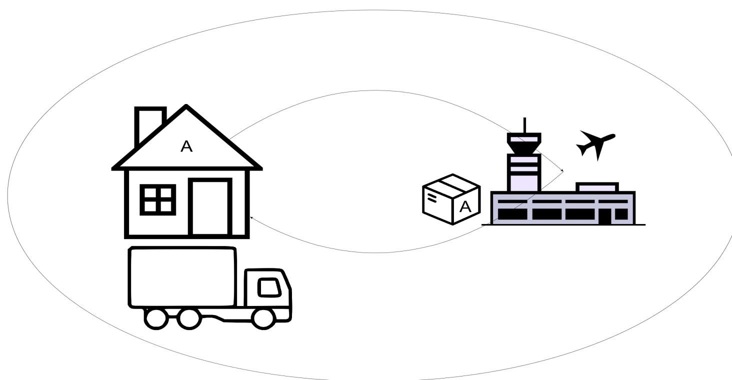

Figure 4.2: Minimal example

Definition 7 (Orderings)
Four different ordering types are defined originally by Hoffmann et al. (2004) and modified by Richter and Westphal (2010):

- **Reasonable Orderings**: between φ and ψ, written as $\varphi \rightarrow_r \psi$, if in every s-plan π where ψ is added at time i and φ is first added at time j, where $i < j$, means that ψ cannot hold true at time m, where $m \in \{i+1, \ldots, j\}$ and is true at some time k with $j \leq k$.

- **Natural Orderings**: between φ and ψ, written as $\varphi \rightarrow_n \psi$, if in each plan φ is true at time j before ψ time i. Meaning: $j < i$
- **Greedy necessary Orderings**: between φ and ψ, written $\varphi \rightarrow_{gn} \psi$, if in each plan φ is true at time i-1 before ψ is first added at time i.
- **Necessary Orderings**: between φ and ψ, written $\varphi \rightarrow_{nec} \psi$, if in each plan φ is true at time i-1 when ψ is added at time i.

A logistics task can have every type of orderings. However, we will explain two orderings a bit more thoroughly: *natural* and *reasonable*. Every landmark in the initial state, where the package is not at its destination, is true before the landmarks of the goal state, which induces a natural ordering. Since we are concerned with the time at which a vehicle must be at a certain position, but in principle, there are no restrictions on the order to which they can travel, we are generally not dealing with natural orderings but with reasonable orderings, if we consider each package separately.
Orderings have different restriction levels, which means, the more restrictive, the stronger an ordering. The most restrictive ordering is the necessary ordering, followed by greedy necessary orderings. Natural orderings are in between reasonable and greedy necessary orderings. Reasonable orderings however are the least restrictive orderings, therefore also the weakest.
All properties, that are not given in the initial state and need explicit load/unload *actions*, are part of every s-plan and are called *action landmarks*. Not to be confused by a fact landmark, corresponding action landmarks are induced by fact landmarks: $LM_a = \bigcup_{f \in F} \{a \in A | \, f \in \mathit{eff(a)}\}$ where F is the (disjunctive) fact landmark. In the example Fig. 4.3, we see that the truck can drive from B to C or from A to C. This can lead to sub-optimal plans because the action *drive* from A to C does not have to be an action landmark. However, we still need to capture the move A to C, therefore introducing *disjunctive action landmarks*. To be clear and avoid confusion, for the generation of landmarks we use *fact* landmarks for the calculation of the heuristic, in chapter 5, we consider *disjunctive action* landmarks.

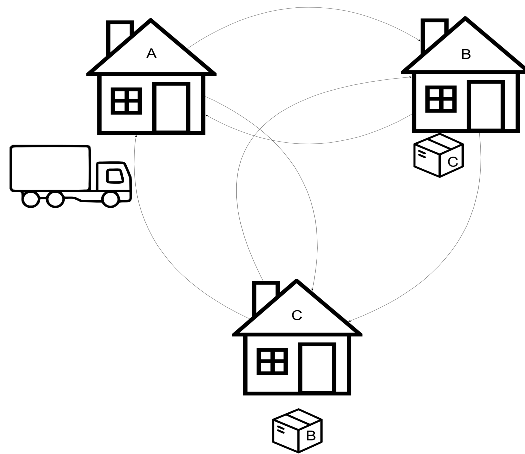

Figure 4.3: Example for action landmarks

Definition 8 (Disjunctive Action Landmarks) As described in Büchner (2020) a disjunctive action landmark of a planning task $\Pi = \langle V, s_0, s_\star, O, C \rangle$ is a non-empty set of actions where at least one is part of the s-plan.
Whenever we talk about landmarks, we will refer to the action landmarks over the fact landmarks.
Both definitions 6 and 7 can be used to induce a landmark graph, a representation of landmarks and their orderings. Following the definition from Büchner (2020):

Definition 9 (Landmark Graph) Let Π be a planning task, s a state of Π. A landmark graph $\mathcal{G} = \langle V, E \rangle$ is a directed graph with a set of vertices V and a set of edges E. There is a vertex in V for every landmark for s. The graph has an edge $\langle l, l' \rangle$ with label t between two vertices l and l' if

- there exists a landmark ordering $l \rightarrow_t l'$ with ordering type t, and
- there is no landmark ordering $l \rightarrow_t l'$ with t' stronger than t.

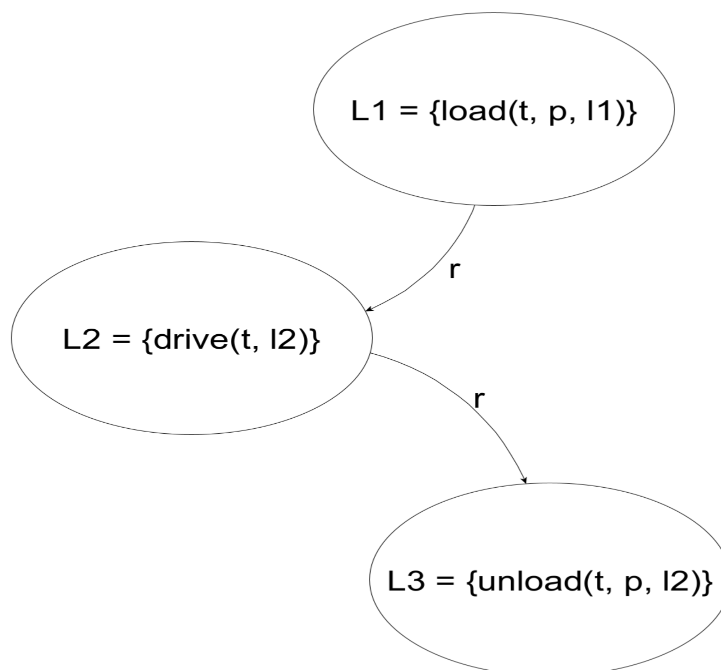

Figure 4.4: A part of an example landmark graph from Fig. 3.2

In the example landmark graph 4.4, we see different landmarks and their orderings. Every single landmark is reasonably connected. The importance of reasonable orderings will be shown in the next chapter, the cyclic landmark heuristic.

5 Cyclic Landmark Heuristic

As we learned quite a lot about landmarks, we can now use them to actually calculate a heuristic. The heuristic, that we want to calculate, is called the cyclic landmark heuristic. It is based on the findings of Paul and Helmert (2016) in "Optimal Solitaire Game Solutions", which was further elaborated to the logistics task (Paul et al., 2017), which is the domain of focus in this thesis.
To explain the cycle heuristic, we will look at the following example:

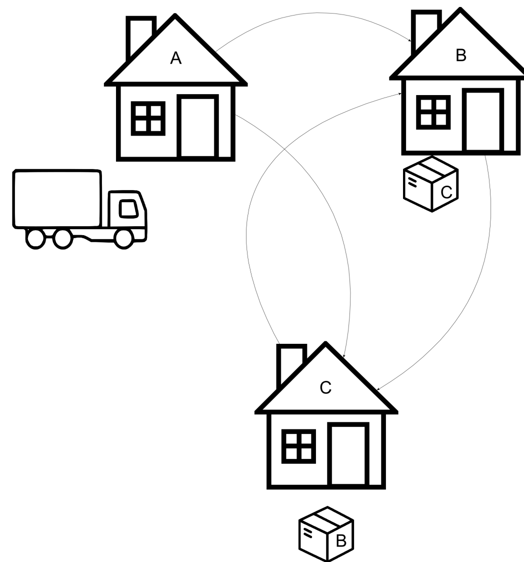

Figure 5.1: A landmark graph with three locations and a truck, where the edges are reasonable orderings

First and foremost we need to understand what a cycle is. In the example 5.1, all locations are connected. A cycle in a graph is a sequence that repeats in our case *actions*. The truck can drive from A to B or C. From A it can either drive to B, pick up package C and drive to C or drive to C, pick up package B and drive to B. Driving from B to C and back would

denote a *cycle*.
In our example, the packages B and C are not at their destination location as we defined in Fig. 3.2. To deliver package B to its destination, the truck has to drive to its origin and pick it up before driving to the destination. The same applies to package C. Noticeably there are reasonable orderings in the shown graph: The truck does not have to be at position B before being at position C, because there are two packages that need to be delivered. By observation, we see that there is a cycle between location B and location C. Therefore the truck has to drive to one of those locations at least twice. This results in an improvement of the heuristic value.
In Büchner et al. (2021a), they define orderings as either *weak* or *strong*. A *reasonable* ordering would correspond to a *weak* ordering if the special case with $j \leq k$ is strict, meaning $j < k$. The cyclic landmark heuristic can be calculated using the operator-counting framework (Pommerening et al., 2014), which solves an Integer Program (short IP) in every state:

$$min \sum_{a \in A} Y_a \, cost(a) \quad s.t. \tag{5.1}$$

$$Y_a \geq 0 \qquad \text{for all } a \in \mathcal{A} \tag{5.2}$$

$$\sum_{a \in L} Y_a \geq 1 \qquad \text{for all } L \in \mathcal{L} \text{ and} \tag{5.3}$$

$$\sum_{L \in \mathcal{L}(c)} \sum_{a \in L} Y_a \geq |\mathcal{L}(c)| + 1 \qquad \text{for all } c \in \mathcal{C} \tag{5.4}$$

The L component of the IP denotes the landmark constraints, whereas C is a set of cycles in a landmark graph $\mathcal{G}$.

6 Landmark Generator

In this chapter, we want to clarify how to generate the logistic landmarks and the corresponding landmark graph as defined in Paul et al. (2017).

6.1 Generating landmarks

Landmarks for the logistics task can be created very specifically. We distinguish between two different cycle heuristics, *the cycle heuristic* h_{cycle} and the integrated cycle heuristic h_{ic}. The only difference is that we add more orderings to the integrated cycle heuristic than to the cycle heuristic. To find the heuristic value of those, landmarks need to be created.

Definition 10 (Truck Landmark) For a logistics task $\langle L, C, P, T, A, city, airport, origin, dest \rangle$, state s and city $c \in C$, the set L_c^{truck} of truck landmarks consists of the locations l that have an ingoing edge in the truck delivery graph D_s^c or that have an outgoing edge and there is no $t \in T$ with $s(t) = l$.

Definition 11 (Airplane landmark) For a logistics task $\langle L, C, P, T, A, city, airport, origin, dest \rangle$ and state s, the set $L^{airplane}$ of airplane landmarks consists of the cities c that have an ingoing edge in the airplane delivery graph D_s^a or that have an outgoing edge and there is no $a \in A$ with $s(a) = airport(c)$.

In an optimal plan, packages should not be loaded and unloaded twice in the same city by different trucks or even the same truck. This is unnecessary and will therefore be eliminated from consideration. We have a list of load/unload actions if the packages are already in the destination city:

- unloaded from an airplane, only if it is in an airplane,
- loaded into a truck if the current position is not the destination and it is not in a truck already, and
- unloaded from a truck if its position is not the destination and it is in a truck.

Checklist: Package in destination city.

There are more actions to consider if the packages are not in the destination city:

- loaded into a truck iff its current position is not in the truck or at an airport,
- unloaded at an airport iff it is still in the wrong city and in a truck,
- loaded into an airplane iff it's not in an airplane,
- unloaded from the airplane in the destination city,
- loaded into a truck if current position is not the destination and it is not in a truck already, and
- unloaded from a truck if its position is not the destination and it is in a truck.

Checklist: Package not in destination city.

An important note is, that a package can only be, as already defined in the logistics domain 3.1, in three specific positions: at a location, in a truck, or in an airplane. Note that a location can either be an airport or a *position* itself.
This can be represented in a landmark graph.

Definition 12 (Landmark Graph of City and Air Space)
For state s of task $\langle L, C, P, T, A, city, airport, origin, dest\rangle$ and city $c \in C$, the landmark graph for c is the directed graph $G^{LM}_{c,s} = (L^{truck}_c, E)$, where E contains an edge $l \rightarrow l'$ if the delivery graph D^c_s contains such an edge and there is no $t \in T$ with $s(t) = l$. The landmark graph $G^{LM}_{air,s}$ for the air space is the directed graph $(L^{airplane}, E)$ where E contains an edge $c \rightarrow c'$ if the airplane delivery graph D^A_s contains such an edge and there is no $a \in A$ with $city(s(a)) = c$.
As already mentioned we distinguish two cyclic heuristics, one with integrated orderings and one without. The above landmark graph corresponds to the normal cycle heuristic. Additionally to all orderings and landmarks above we find the following orderings and landmark graph.

Definition 13 (Integrated Landmark Graph)
For state s of task $\langle L, C, P, T, A, city, airport, origin, dest\rangle$ the integrated landmark graph $G^{LM}_s = (V, E)$ is the directed graph, where

- $V = L^{airplane} \cup \bigcup_{c \in C} L^{truck}_c$ consists of all truck and airplane landmarks, and
- E contains all edges from the landmark graphs for all cities and the air space plus the following edges for each package p with $city(pos_s(p)) = c$ and $city(dest(p)) = d \neq c$:
  1. if there is no $t \in T$ with $s(t) = pos_s(p)$ and neither $pos_s(p)$ nor $dest(p)$ is an airport, there is an edge $pos_s(p) \rightarrow dest(p)$;
  2. if there is no $t \in T$ with $s(t) = pos_s(p)$ and $pos_s(p)$ is not an airport, there is an edge $pos_s(p) \rightarrow d$;

3. if neither $pos_s(p)$ nor $dest(p)$ is an airport, there is an edge $airport(c) \to dest(p)$;
4. if $pos_s(p)$ is not an airport, there is an edge $airport(c) \to d$;
5. if there is no $a \in A$ with $s(a) = airport(c)$ and $dest(p)$ is not an airport, there is an edge $c \to dest(p)$;
6. if $dest(p)$ is not an airport, there is an edge $d \to dest(p)$.

To clarify we consider the following example:

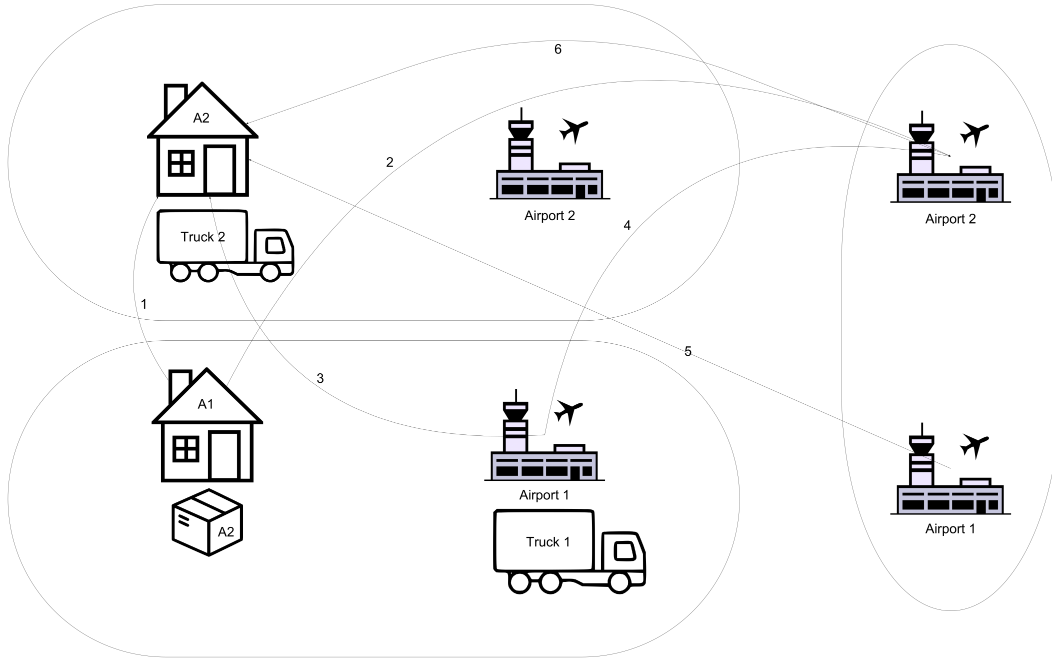

Figure 6.1: The integrated landmark graph 6.1 explained, with numbers 1-6 corresponding to the edges that are added

All additions of landmarks, from the truck landmark, airplane landmark, and loading/unloading landmarks are easily added with a pre-processing unit, we devised.

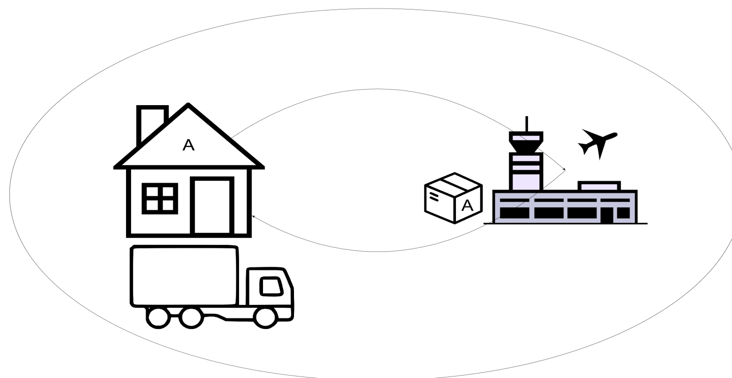

Figure 6.2: Package in the destination city

Landmarks are added based on the current package position and city location. If the current city is already the destination city of the package, we can use the checklist 6.1 to verify which

load/unload landmarks should be added. The *drive* operator should only be added if the position of the truck does not correspond to the package position. In the example 6.2, the following landmarks are added:

1. drive truck to the airport
2. load package A into a truck
3. drive truck to position A
4. unload package A from a truck

Consider another example where the package is not in the destination city. We need several more landmarks to accomplish the successful transportation of the package to its destination. We again verify load/unload landmarks with the provided checklist 6.1. The next example 6.3 shows a situation where the package is not in the destination city.

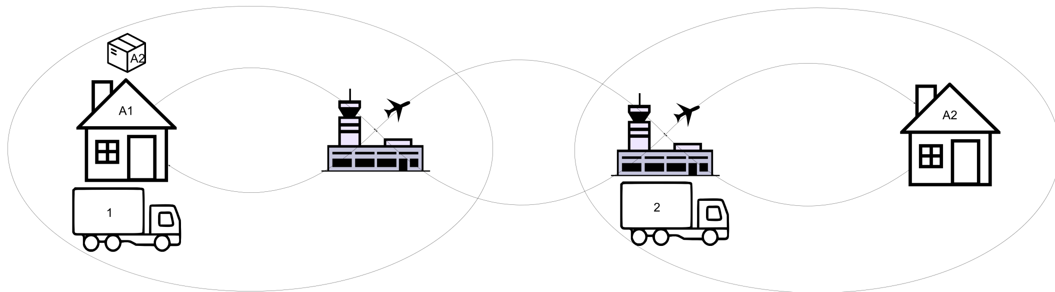

Figure 6.3: Package not in the destination city

Note that the airplane which is not clearly visible in this example is located at the left airport. As defined in definitions 10 and 11, the delivery graph decides whether a landmark is added for trucks and airplanes or not. For example 6.3, the following landmarks are added:

1. load package A2 into truck 1.
2. drive truck 1 to the airport in origin city.
3. unload package A2.
4. load package A2 into an airplane.
5. fly to the destination city.
6. unload package A2 at the airport in destination city.
7. load package A2 into truck 2.
8. drive truck 2 to position A2.
9. unload package A2 from truck 2 at destination position.

7 Experimental Evaluation

In the experiments we consider the logistics domain logistics00, logistics98 from the IPC benchmarks set, and additional logistical benchmarks. The difference between those benchmark sets is that logistics98 includes multiple trucks in one city, whereas the others are the same. All experiments are conducted on the grid at the University of Basel with a 2×10 Core Intel Xeon Silver 4114 2.2 GHz Processor. Calculations are performed with the Downward Lab (Seipp et al., 2017). The task may run up to 30 minutes before running out of time and the memory limit is set to 3.5 GB. The heuristics need a linear program solver which is provided by IBM CPLEX version 20.10 (IBM, 2021). We want to compare our findings with the ones provided by Paul et al. (2017). However we did not include multiple vehicle simplification, therefore this will be left out. The search algorithm will be $A^\star$ with an admissible heuristic, the cyclic landmark heuristic. In addition to the logistics-specific landmark generator that we created, we will also add the domain-independent landmark generator LM-RHW (Richter et al., 2008). The planner of choice is the Fast Downward planner (Helmert, 2006) which is implemented in C++. We pick up on previous work from Büchner et al. (2021b) and generate landmarks based on the criteria, we are provided with from Paul et al. (2017). Büchner et al. (2021a) wrote in his paper, that current landmark generators are not finding sufficient landmarks with domain-independent approaches. Therefore, we created a landmark generator for the logistics-specific task, as well as a foundation for more domain-specific landmark generators. We want to share results on our findings and compare values with Paul et al. (2017). They used a different planner than our planner of choice, but we want still to compare key attributes in this chapter. Some plots that are shown are logarithmic plots. Meaning, all values 10^{-1} are considered as zero, etc. We will divide the benchmarks set in this chapter. Logistics00 and the additional benchmark set will be plotted together since they are similar. In all plots, the diagonal denotes values that are identical in the compared configurations.

7.1 Initial heuristic value

We distinguish between the integrated cyclic landmark heuristic and the cyclic landmark heuristic, as well as for LM-RHW and the integrated cyclic landmark heuristic.

7.1.1 Logistics00 and additional benchmark

In Fig. 7.1, we see the difference between the initial heuristic value of the integrated cyclic landmark heuristic, denoted as the y-axis, and the cyclic landmark heuristic, as the x-axis.

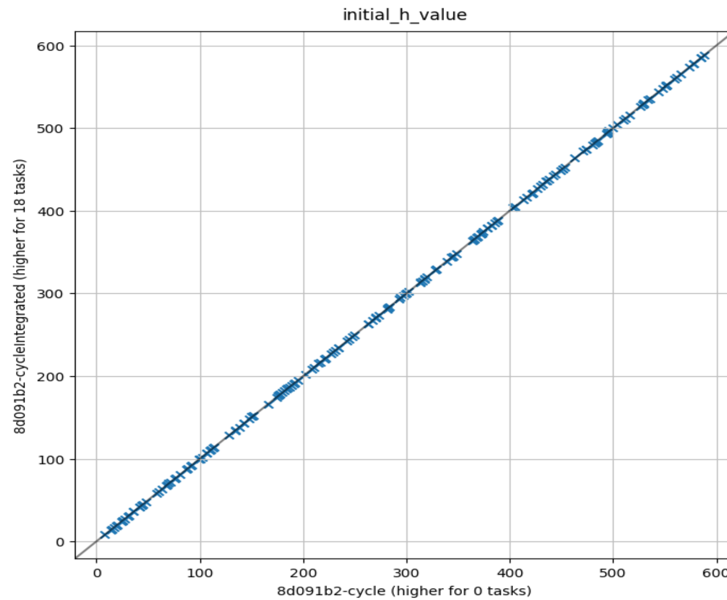

Figure 7.1: Initial heuristic value for cycle vs integrated cycle

The difference is mostly not visible because, in only 18 tasks, the integrated cyclic landmark heuristic finds a better heuristic value for the problem. This can be explained by the definition in chapter 6. The integrated landmark graph contains more orderings than the non-integrated landmark graph. There might be a cyclic dependency, which would improve the heuristic value.

Much more interesting is the comparison Fig. 7.2 between RHW and the integrated cyclic landmark graph. We notice immediately that the latter finds more landmarks for the initial state than RHW. This shows that domain-specific landmark generators yield better results than domain-independent generators since the latter finds fewer landmarks. In the plot Fig. 7.2, two lines emerge. There is an overlapping area with a gap of different initial heuristic values. Since we did not have the time to experiment more and examine this area, we will pick it up again in chapter 8.1, for future work.

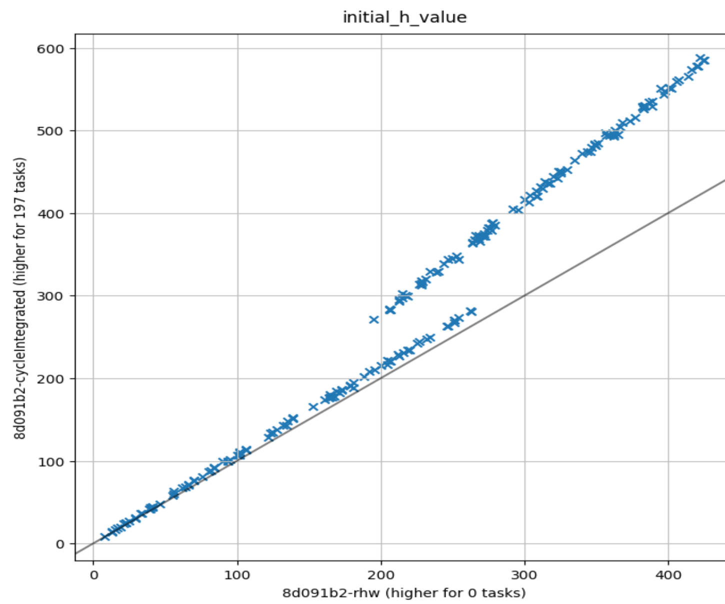

Figure 7.2: Initial heuristic value for RHW vs integrated cycle

7.1.2 Logistics98

Different from logistics00 and the additional benchmark set, we see no difference in the initial heuristic value. This is quite interesting since normally this would mean the orderings from the integrated version are not accounted for, which would state that there are no more cyclic dependencies for different packages.

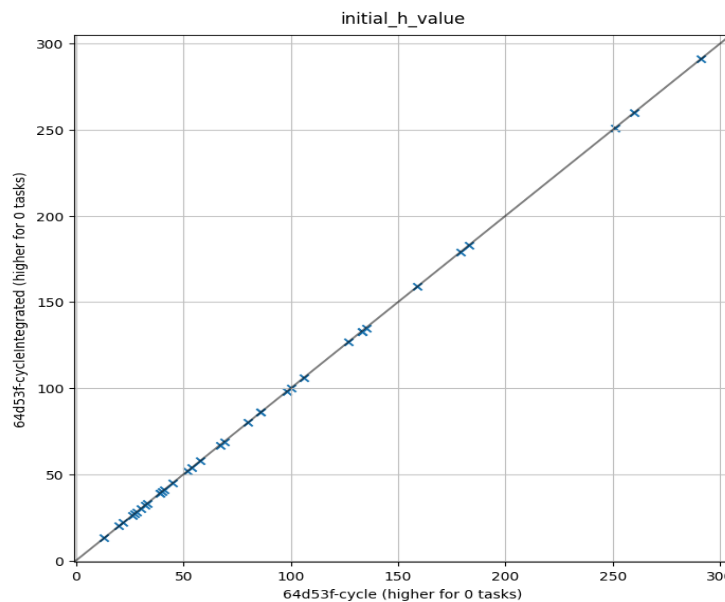

Figure 7.3: Initial heuristic value for cycle vs integrated cycle logistics98

In comparison to RHW however, we yield much better results. Nearly all of the heuristic values are higher with our implementation. We notice again that RHW loses some landmarks which are essential and therefore possibly does not solve as many tasks as it should. Since

35 problem tasks are not that many to solve, when considering possibly more logistics98 problems, there might be again two lines in Fig. 7.4 and the overlapping area, which would be interesting to look at.

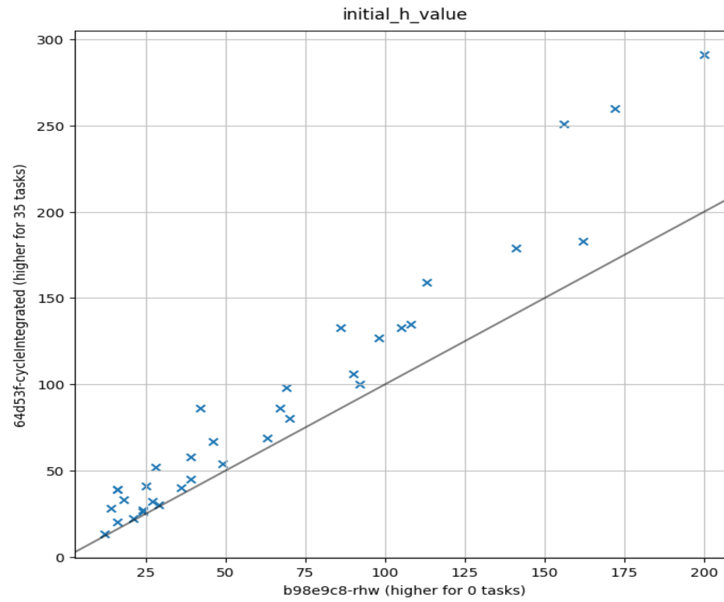

Figure 7.4: Initial heuristic value for RHW vs integrated cycle logistics98

7.2 Total Time

In this section, we want to discuss the total time we used to solve a problem. All experiments had a run time of 30 minutes. In all plots, measured in seconds, points in the top right corner are tasks that are not solved due to them being well over the 1800s.

7.2.1 Logistics00 and additional benchmark

The search time is more or less the same for the integrated and the normal version. In 15 tasks the integrated version is faster than the non-integrated version. This may come from the tasks which had a higher heuristic value, whereas the A* algorithm traverses the state space differently. However, the non-integrated version is faster in 18 tasks. The cause of this might be the extra checks we need to make where the better heuristic guidance doesn't compensate or the heuristic is equal for the integrated and normal version, making the additional checks obsolete.

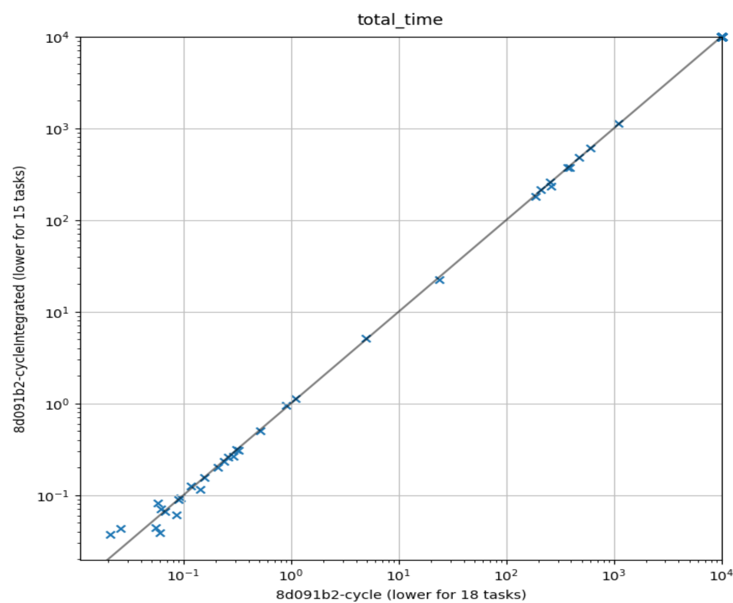

Figure 7.5: Total time for cycle vs integrated cycle

Again the comparison between RHW and the integrated landmark graph is more interesting. In every single task, our implementation, which terminated, is faster and solves more tasks than RHW. The specificity of the generation of landmarks has a visible impact on the search algorithm.

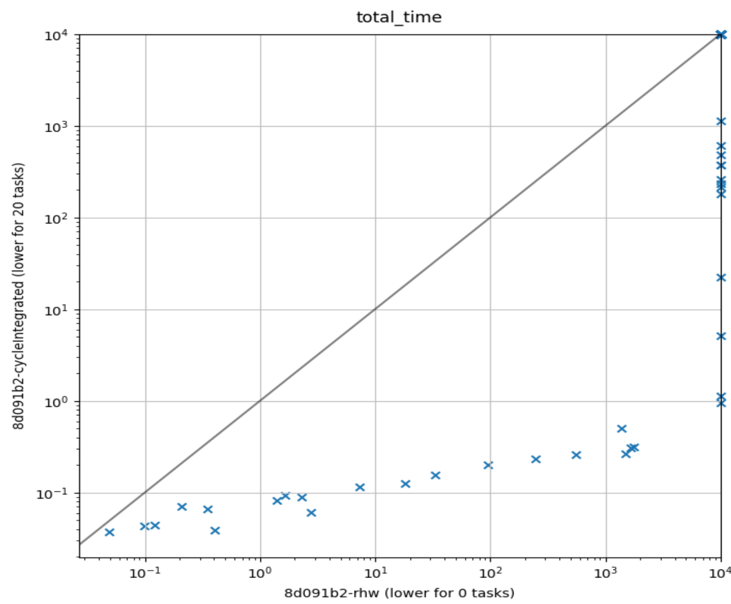

Figure 7.6: Total time for RHW vs integrated cycle

7.2.2 Logistics98

The total run time for logistics98 is not much different from logistics00 and the additional benchmark set. What might be the reason for this, is the noise on the grid. Since both the normal and the integrated versions have the same heuristic value for these experiments, we cannot make more assumptions.

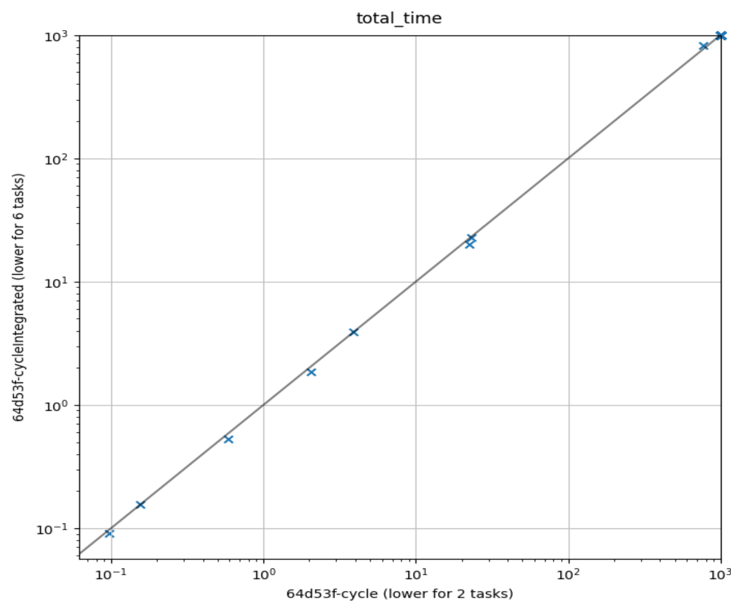

Figure 7.7: Total time for cycle vs integrated cycle logistics98

The difference is seen however in the comparison between RHW and the integrated version. Noticeably our implementation is faster for all tasks that terminate, RHW is running into a search timeout faster than ours does. However, since the coverage of the tasks for the

logistics98 domain is quite low, there might be some issues to address.

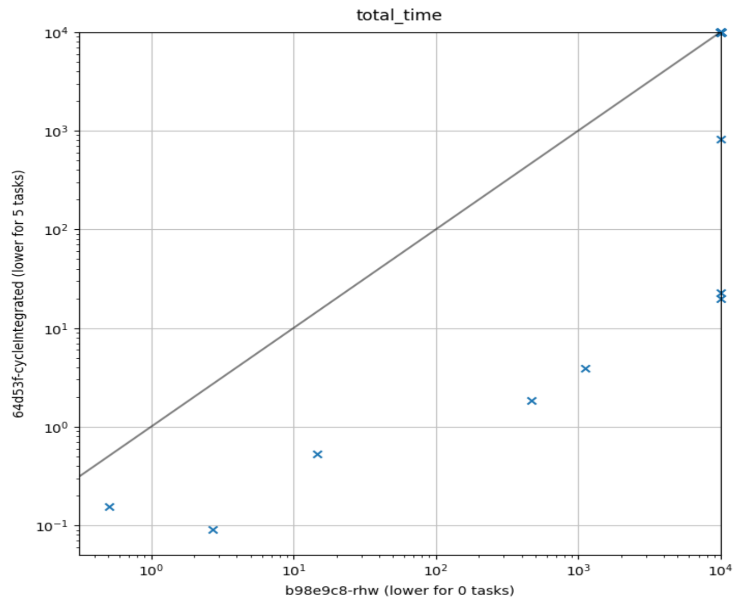

Figure 7.8: Total time for RHW vs integrated cycle logistics98

7.3 Expansions until last jump

In this section, we want to discuss the *expansions until last jump*. If the value of the jumps is 10^{-1}, this means, that we found the perfect heuristic for these specific problem tasks.

7.3.1 Logistics00 and additional benchmark

Interestingly enough all expansions are the same in both integrated and cyclic landmark graphs, but we cannot make any assumptions with these results.
On the other hand, we see a difference between RHW and the integrated cyclic landmark heuristic. We find that the integrated cyclic landmark heuristic yields much better results than RHW. Most of the tasks have zero jumps after the last expansion, meaning we receive the perfect heuristic for these problem tasks. We notice that RHW expands more states than we do before finding a solution. Most of the tasks we solve are close to the perfect heuristic whereas RHW expands too many states, possibly meaning its heuristic values are less accurate, it does not find as many landmarks and orderings as we do. There might be cyclic dependencies in the graph, which are not detected by RHW, which leads to lower heuristic estimates and ultimately requires more state expansions.

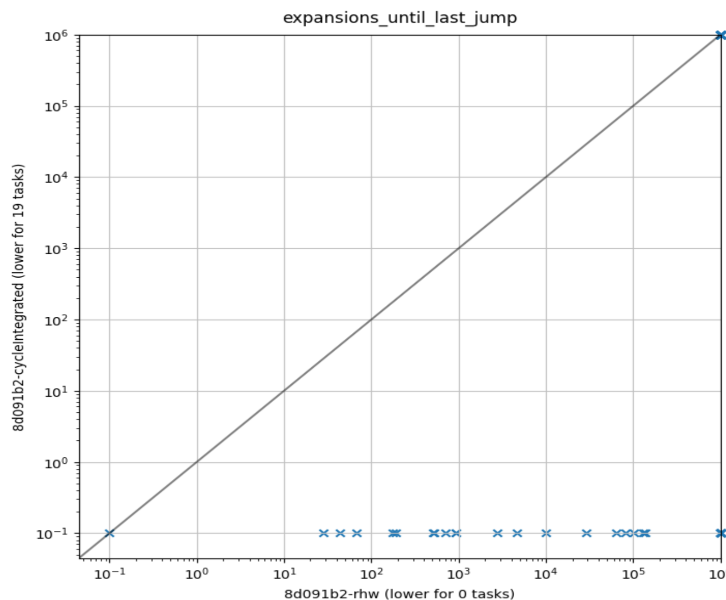

Figure 7.9: Expansions until last jump RHW vs integrated cycle

7.3.2 Logistics98

Again we notice no difference between the normal version and the integrated version, which is again surprising. However, we cannot make any assumptions, since possibly both heuristics are identical.
We can see in Fig. 7.11 that the integrated version expands fewer states than RHW again. Even though achieving the perfect heuristic for the initial state sometimes, the expansions until the last jump does not have to be zero. Since we need to consider all states with perfect

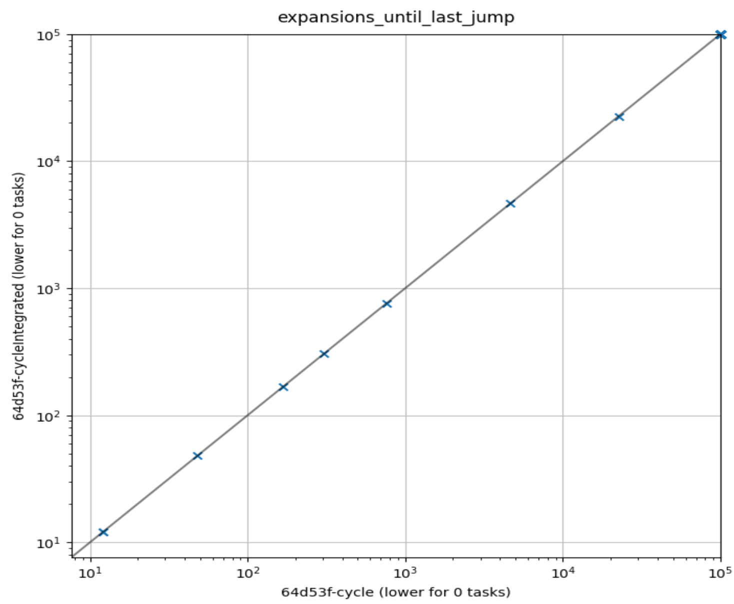

Figure 7.10: Expansions until last jump for cycle vs integrated cycle logistics98

heuristic value, this may lead to expansions, even with perfect heuristic in the initial state. Correct

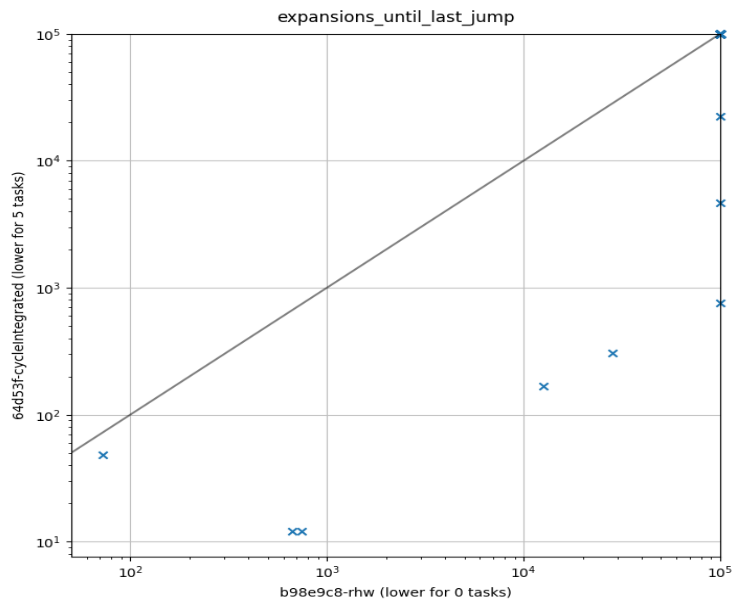

Figure 7.11: Expansions until last jump RHW vs integrated cycle logistics98

7.4 Memory

In this section, we want to talk about the memory usage of our implementation and RHW.

7.4.1 Logistics00 and additional benchmark

The comparison between the normal landmark graph and the integrated cycle landmark graph does not show much. There are several tasks where the memory is lower on the normal version than the integrated.

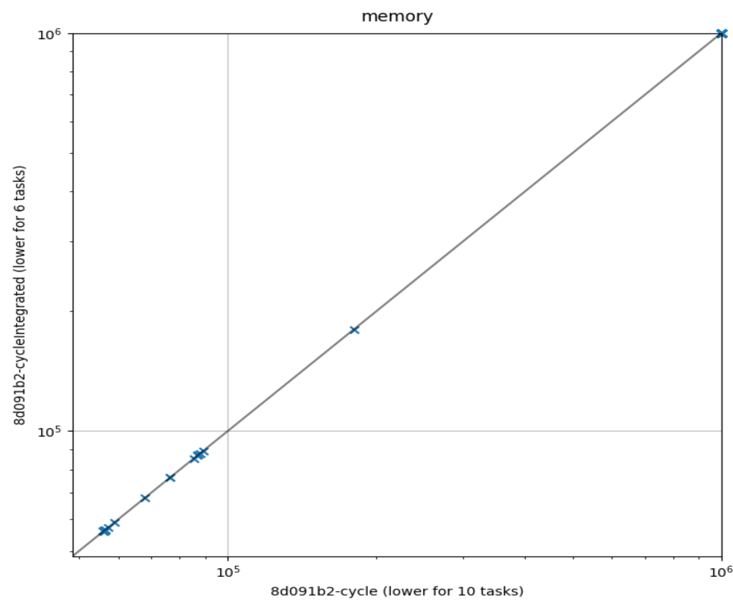

Figure 7.12: Memory comparison cycle vs integrated cycle

The comparison between the integrated landmark graph and RHW is more interesting. The memory bound is for all tasks either the same or better for the integrated. RHW runs quite quickly into the memory limit due to expanding more states, whereas tasks are still solved in the integrated version. This can be seen in plot Fig. 7.13.

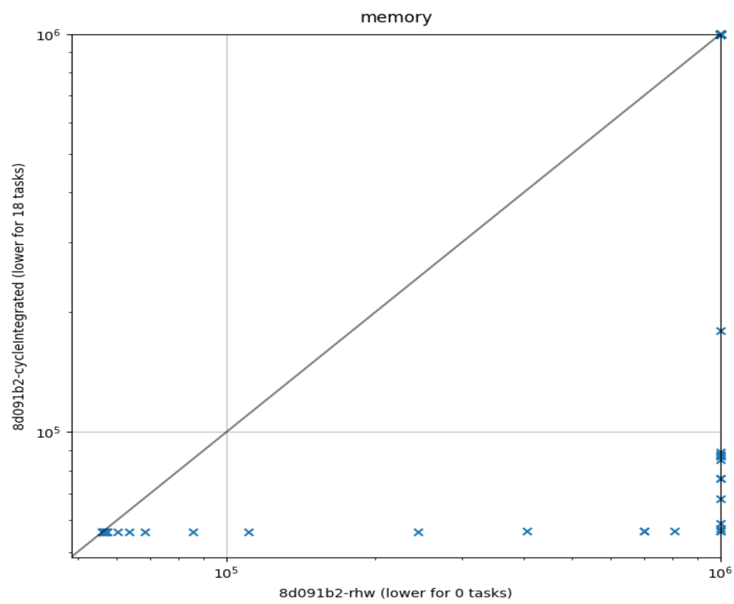

Figure 7.13: Memory comparison RHW vs integrated cycle

7.4.2 Logistics98

The memory comparison between the non-integrated and the integrated version is different from the other benchmark sets, the latter being lower for only 3 tasks. Possibly due to expanding fewer states than the normal version

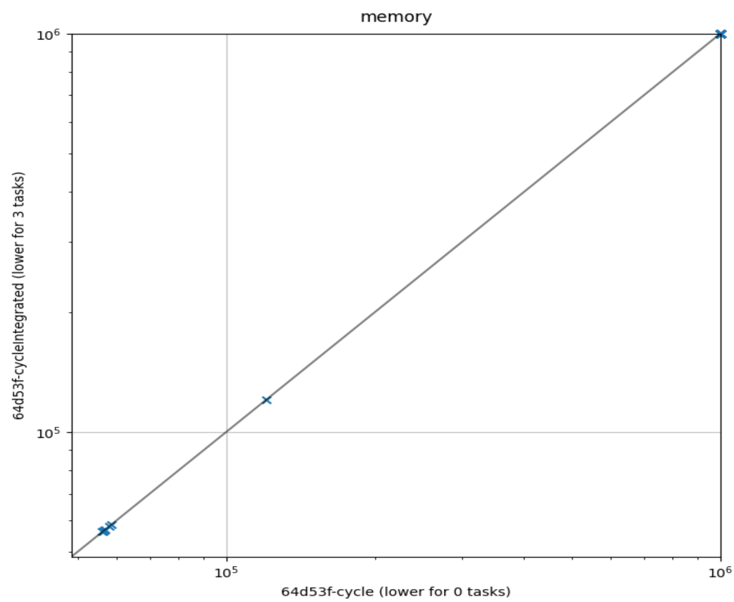

Figure 7.14: Memory comparison cycle vs integrated cycle logistics98

Nonetheless, there is a difference in memory to RHW. For 5 tasks, the integrated version is lower due to the expanded states.

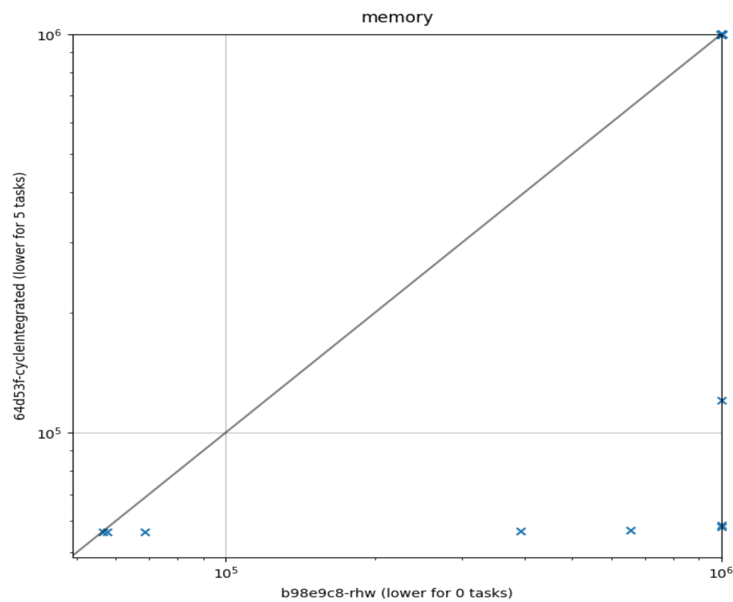

Figure 7.15: Memory comparison RHW vs integrated cycle logistics98

7.5 Comparison to Paul et al. (2017)

When comparing our results to the one from the paper, we notice for logistics98 that something is not as expected. We expand too many states even though finding the perfect heuristic sometimes. Both heuristics, the cyclic landmark heuristic, and the integrated version, yield identical results for the initial heuristic value, which is not the case in the results from Paul et al. (2017). In logistics00, we have no comparison to the paper. However, for the additional benchmark sets, we do. In some problem files from our implementation, the heuristic values are identical for both versions, however, in Paul et al. (2017) there are differences. Sometimes we yield better results and sometimes worse than the results in the paper. Since we do not know the cause, this could also be looked at as future work.

8 Conclusion

In this thesis, we generated landmarks for a domain-specific task, the logistics task. We noticed that domain-specific landmark generators are better suited to solve domain-specific tasks than domain-independent landmark generators. The number of landmarks found, and the corresponding orderings between them, yield higher in our landmark generator.
We first looked at the representation of a logistics planning task and which vehicles we use to transport a package to its given destination. We then discussed landmarks, actions that need to occur at least once in every plan. We discussed that logistics tasks can include cycles and used them for the cyclic landmark heuristic. We noticed that by introducing cycles, a key location has to be visited at least twice which yields better results for the heuristic estimate. We then looked at landmarks for the logistics tasks, how they are generated, and the creation of the landmark graph.

8.1 Future Work

For future work, we have a few things to address. It would be exciting to see the difference between the RHW landmarks and the domain-specific landmarks to improve the domain-independent methods, ultimately also improving the results of other domains.
The overlapping area in Section 7.1 is very interesting to look at, as it is unknown why there is such a *jump* in the plots. Addressing this in future work could be very interesting.
Also during the thesis and the implementation, we came to realize that landmarks in the initial state, that are true and need further actions to reach the goal state, probably need some more research. During the implementation, there were some issues to address with the initial state landmarks, which should have been accounted for, however, they were not.
In the future *state* of this world, you should never worry about your package being efficiently delivered to your home!

A
Appendix

We discovered ways to generate landmarks and their corresponding landmark graph. An important step before assembling all the data and generating the landmark graph was preprocessing. As the data can be hard to work with sometimes, this contribution is important to remodel, redesign and rethink logistics-specific tasks or in general *domain-specific* landmark generators. Therefore we want to introduce an essential implementation part of the thesis.

A.1 Benchmarks and their structure

We work with the IPC benchmarks logistics98 and logistics00, as well as an additional benchmark that is similar to the logistics00. The difference between those two benchmark sets is that logistics98 is more complex to solve than logistics00 since there are more than two locations and there are more than a single truck in a city. The logistics00 set does only have two locations and a single truck per city. What they have in common is the number of airplanes. Not by actual amount but both sets use more than one airplane to fly between cities.

A.2 Information Assembling

In the Fast Downward planner from Helmert (2006), information is stored in different kinds of structures. These structures are used to assemble data and store the data in hash maps for constant lookup times. From a task proxy or state, we can retrieve fact pairs as defined in chapter 4. Every fact pair can be divided into atoms, which cannot be further divided. As an example, a fact (truck at position 2) can be divided into truck or position 2. As this data is not unique (variable $\rightarrow$ value), this bears some problems. For example, position 1 in city 1 and position 2 in city 2 have the same value, thus making it harder to code. To ensure uniqueness, we introduce a structure to map various positions to a unique value.

MapperPair The MapperPair is quite intuitive. It maps one value to another. It is used to ensure the uniqueness of the data and to easily create facts on the go. Facts are created

by using the lookup maps. A way to create fact pairs is shown in the following algorithm:

Algorithm 1 Creating fact pairs from mapper pairs

```
 1: function ToFactPair(MapperPair)
 2:     variable_iterator ← locationToVariableIndex        ▷ Get the iterator of the hashmap
 3:     assert variable_iterator ≠ end of locationToVariableIndex
 4:     factVariable ← variable from variableIterator                  ▷ Get fact variable
 5:     value_iterator ← valueOfLocation.from(factVariable)    ▷ Get corresponding values
 6:     factValue ← -1
 7:     for i ∈ value_iterator → locations do
 8:         if value is searched value then
 9:             factValue ← i
10:             break
11:         end if
12:         assert factValue ≠ -1                                          ▷ Not found
13:         return ⟨ factVariable, factValue ⟩        ▷ This is the fact pair we searched for
14:     end for
15: end function
```

This way we ensure to get the right fact pairs for the creation of the landmarks.

City Info We now have a foundation for the creation of fact pairs. However, we still need information about the current city, their locations, and their trucks. We could use the state and filter information from there. An easier approach to get information about the current city was to parse all relevant data into a struct, which ultimately was called CityInfo. CityInfo consists of four crucial elements:

- A list of positions
- An airport (since a city can only have one airport in these problem files)
- One truck or (if more trucks available) a list of trucks, and
- the city itself

All CityInfos will be stored in a vector, creating a "map". Note that an airplane can leave cities and fly between different airports, and therefore will not be stored in the CityInfo.
To retrieve information, we will use the following method:

Algorithm 2 Create the city information

```
1:  function GetCityInfoPtr(Location)
2:      if hasType City then
3:          search the city in vector
4:          assert city found
5:          return copyOfCity
6:      else if hasType Airport then
7:          search the city with corresponding airport in vector
8:          assert city found
9:          return copyOfCity
10:     else if hasType Truck then
11:         search the city in vector and iterate through all trucks ▷ In logistics00 we do not
        have to iterate through all trucks since there is only one truck in each city
12:         assert city found
13:         return copyOfCity
14:     else if hasType Position then
15:         search the city in vector and iterate through all positions
16:         assert city found
17:         return copyOfCity
18:     else
19:         assert false
20:     end if
21: end function
22: function GetCityInfo(Location)          ▷ This function will create a copy of the
    information
23:     cityInfo ← GetCityInfoPtr(Location)
24:     if cityInfo ≠ nullptr then
25:         return cityInfo
26:     end if
27:     return wrongCity                     ▷ this will lead into a crash
28: end function
```

This framework we provide is an important step to assemble data in a comfortable way. If there is any need for further explanation, check the github repository.

Bibliography

Clemens Büchner. Generalization of cycle-covering heuristics, 2020.

Clemens Büchner, Thomas Keller, and Malte Helmert. Exploiting cyclic dependencies in landmark heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 65–73, 2021a.

Clemens Büchner, Thomas Keller, and Malte Helmert. Code, benchmarks and experiment data for the ICAPS 2021 paper "Exploiting Cyclic Dependencies in Landmark Heuristics", 2021b. URL https://doi.org/10.5281/zenodo.4604735.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

IBM. Ibm cplex studio, 2021. URL https://www.ibm.com/products/ilog-cplex-optimization-studio.

Drew M. McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2): 35, Jun. 2000. doi: 10.1609/aimag.v21i2.1506. URL https://ojs.aaai.org/index.php/aimagazine/article/view/1506.

Gerald Paul and Malte Helmert. Optimal solitaire game solutions using a search and deadlock analysis. *Heuristics and Search for Domain-independent Planning (HSDIP)*, page 52, 2016.

Gerald Paul, Gabriele Röger, Thomas Keller, and Malte Helmert. Optimal solutions to large logistics planning domain problems. In *Tenth Annual Symposium on Combinatorial Search*, 2017.

Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. Lp-based heuristics for cost-optimal planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 24:226–234, 2014. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13621.

Silvia Richter. *Landmark-Based Heuristics and Search Control for Automated Planning.* PhD thesis, Citeseer, 2010.

Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *AAAI*, volume 8, pages 975–982, 2008.

Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward lab, 2017. URL https://doi.org/10.5281/zenodo.790461.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Günes Aydin

Matriculation number — Matrikelnummer
2018-058-636

Title of work — Titel der Arbeit
Evaluating the Cyclic Landmark Heuristic with a Logistics-specific Landmark Generator

Type of work — Typ der Arbeit
Bachelor thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognized scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, September 12, 2021

Signature — Unterschrift