PAR - MAI
Goal Stack Planner
for
Train Management

José A. Magaña Mesa, Rodrigo Weffer
PAR | Second activity | December 2013

INTRODUCTION

In this work we developed a solution for an intelligent railroad station. In it there is a finite number of coupled wagons and one locomotive. The system must deliver to the locomotive a good plan to position the wagons so they can be loaded and unloaded at the station and finish in the right order for departure.

A Linear Goal Stack Planner was developed to address the problem automatically; it must be able to solve every situation that arises with the given constraints. This developed scheme had to be efficient and flexible enough to find an approximate to the best solution for the problem while adapting to any change it might receive in the initial state, such as more railways in the station.

In the specification of the problem we consider two things. the predicates and the operators. The Predicates are a set of boolean conditions that will apply in different stages of the problem and are used to express the subgoals to be met. The Operators are a set of actions that change the state of the world. The Operators in their turn are comprised of three things: The Preconditions which must be satisfied so the Operator can be applied, an Add list, a list of predicates the operator causes to become True and the Delete list, a list of predicates the operator causes to become False.

In our system, the problem to solve is read from a text file. This file contains the identifier of wagons to consider, and the initial state and the final state as they are identified by the satisfied predicates in those states. These states will be analyzed and in the end, the system will return a List of Operators which is known as the Plan. This is the order in which the Operators must be applied to solve the problem.

The station has a maximum number of railways set to default on 3, but it can be indicated otherwise indicating so in the Problem definition with an optional value (Max_railways=4).

To make graphical examples we will use the notation used by the system to log visually, we will explain the example in Figure 1. Each “|-” is the representation of a railway and they go from left to right. The letters notate the wagons identified by the letter, parenthesis around it mean that the wagon is loaded and the lack of them that is empty. The “Loc[K]” symbol is the state of the locomotive, the letter K notates that the locomotive is currently towing the wagon “K”, if the locomotive was free, the notation would be “Loc[]”.

```
|-(A)-B-(F)
|-E-(D)-(C)-G
|-(H)-I-J
Loc[K]
```

Figure 1. example of the graphical notation.

ANALYSIS

As we have stated the problem will be addressed through a linear goal stack planner. This means that we must find a way to satisfy a global goal by first accomplishing a set of subgoals in a sequential manner. The fact that we must maintain the planner linear limits us in many ways since we cannot do a multiple solution search and once a step is taken it should not be undone by future applied operators. To face this issue then an approach must be selected in a way that the state space of solutions is searched as optimally as possible.

The linear goal stack planner will only make use of a Stack structure where the algorithm keeps track of the Goals and the Operators to apply in order and the current State, comprised of the Predicates currently marked as true.

The order in which the Predicates and the Operators are added to the Stack is the reversed order in which the goals will be attempted to make become True or in which actions to satisfy these Goals will be taken. To optimize the order of the elements in the Stack is to optimize the Plan.

In our particular problem we could identify many situations that would make the search of our path in the state space less than optimal. Part of the problem was to identify these situations and avoid them successfully while maintaining the proposed approach, constraints and given list of operators. A few of the recognized problems were:

- The plan could be caught in a loop by a sequence of operators where in one state would satisfy a subgoal, evaluate the new state of the world and have a new subgoal that would lead to a previously visited state. Since there is no stochastic element in our solution and the world is not dynamic this loop would be repeated infinitely.
- An inappropriate sequence of operators could detour the search for the solution through a far from optimal path.
- If no priority is established over the subgoals to accomplish it could occur a situation where a subgoal undoes another previous subgoal, and the solver would have to satisfy it again. This is known as the Sussman Anomaly[1].

To solve effectively this possible list of situations, we had to consider manipulating our planning search space. We began by analyzing the operators and predicates to limit the strategies we could apply to given states. From this analysis we extracted a set of heuristics and rules that reduced the number of options the algorithm considered at every state. These heuristics could be split into the following categories: Operator groups, Wagon manipulation and Divide and Conquer.

Operator groups

We subgrouped the operators into 3 different types::

- Take operators: Detach and Couple, that having the Locomotive free will take a Wagon.
- Drop operators: Attach and Park, that having a Wagon in the Locomotive will drop it.
- Load operators: Load and Unload, that will put or remove goods from a Wagon

This subgrouping allowed us to establish rules as to the order of possible actions reducing our state space to search. Some of these rules are:

- Two Drop operators cannot be executed sequentially, it is required to have a Take operator in between.
- The complementary is also true and two Take operators cannot be executed in a row, it's required to have a Drop operator in between.
- As a consequence Drop and Take operators will have to be executed in an alternate order.
- For simplicity, the Load Operators have been considered that can only be executed when the Locomotive is Free.

This assumption helped the system to be smarter and efficient. The search space for an adequate operator is reduced significantly to a set of helpful operators.

Wagon Manipulation

The wagon manipulation category is a set of heuristics that allowed to approach the handling of the railway wagons in an intelligent manner. These are applicable to which wagon should the next be the target of the next operator, making it easier to establish an efficient sequence of actions, avoiding the looping situation previously identified as one of the potential problems of our linear goal stack planner.

There are four clear simplifications that can be applied in our search of the state space.

- If a Take operator is executed for a wagon (as first parameter), the Drop operator will have as first parameter that same wagon held by the Locomotive. This allows us to have a logical sequence in the plan and to ensure consistency in the effects taking place in the world.
- If a Take operator is executed from one of the railways, the consecutive Drop operator will be applied on another railway different from the original to avoid returning to a visited state.
- When the goal is to to load or unload a given wagon, and a Take operator is applied to another wagon in order to clear a way to the station, the plan will avoid executing the Drop operation in front of the wagon pertaining the goal.

- For the planning algorithm doesn't make sense to apply a Take Operator to the same wagon twice in a consecutive manner. This heuristic was identified as a good way to avoid direct loops in our plan and to reduce the search space.

Divide and Conquer

To approach the plan as optimally as possible we endowed the system with a Divide and Conquer paradigm. We identified several stages of the solving process according to the predicates satisfied in the initial state, the ones on the final state and the difference between them. From this separation we extracted the following stages, these stages are generalizations of the order that certain Goals will be added to the Stack and they are listed in the order they occur in the algorithm. The split of the final state into these sub-objectives or intermediate steps guarantees that the final state will be achieved without inefficiencies (as far as can be expected from a Linear Planner).

Initial Load Stage: These stage where it takes place the resolution of the Load and Unload goals that can be satisfied without making any change of position among the wagons.

Intermediate Load Stage: On this stage the system creates the part of the plan to satisfy load and unload goals on wagons that do not comply with the ON-STATION predicate on the initial or the final state.

For example, in the problem stated in the Figure ZZZ, the letter D changes from satisfying LOADED(D) in the Initial State to satisfying EMPTY(D) in the Final State. Since the operator UNLOAD(D) can't be applied in any of these two states, a part of the plan must be dedicated to achieve this Predicate.

Initial State:
|-A-B-F
|-E-(D)-C
|-H-I-G
Loc[]

Final State:
|-C-G
|-B-D-A-I
|-E-F-H
Loc[]

The objective is to move the wagons upon which the Load predicates must be satisfied (in the figure, the wagon D) to an empty railway so that the ON-STATION(D) precondition to the operator UNLOAD(D) can be fulfilled. This carries a problem of its own, if no railway is already empty, how should the system select the railway to be emptied? So another heuristic is used to solve this issue: The railway selected should be the shortest one, in order to move the minimum number of wagons. In the case of a tie, the competing railways will be checked position by position until we find a wagon that is not in a position it should occupy in the final state. This checkup allows us to avoid dismantling a partly correctly ordered railway. Since the occasion could arise where a railway must be dismantled, and others will be shuffled in order to free A (to satisfy the precondition FREE(A) to apply any Take operator), it makes no sense to consider the subgoals related to a wagon position until this stage is finished.

Move Stage: This stage will address the part of the plan that seeks to satisfy the goals related to the wagons' positions. that is to position the wagons so that the ON-STATION, IN-FRONT-OF, and FREE predicates in the goal are satisfied and not worry itself with which wagons are LOADED and which are empty.

This stage is again subdivided into other problems. How should the plan select the order of the goals to be satisfied? Should it try to solve a goal as it was given in the problem's final state or try to choose a smarter way?

This problem was solved by splitting the problem into smaller problems of ordering one railway at a time. Concentrating the effort on one railway at the time will avoid undoing already (purposely) achieved states guaranteeing that the algorithm will finish successfully. So the algorithm divides the subgoals by railway, solving first the railway with the largest number of wagons in the final state. This criteria is the opposite to the selection of a railway to empty when a railway must be made available.

To visualize the following example we will focus only on the Predicates related to moving the wagons. Given an Initial State and a Final State such as:

Initial State: |-A-B-F |-E-D-C-G |-H-J-K Loc[]	Final State |-G-C-J-F |-B-D-A |-E-K-H Loc[]

A unintelligent implementation of the Plan would create an initial Stack of such as the one shown in the table ZZZ, where the railways are not completed in order, therefore having to undo previously solved Goals. According to that given stack, the Plan would attempt to satisfy IN-FRONT-OF(D,B), before ON-STATION(B) is accomplished. So in the future would have to decouple D and B, in order to manage ON-STATION(B).

Final State	Stack

|-G-C-J-F
|-B-D-A
|-E-K-H
Loc[]
	IN-FRONT-OF(D,B)
	IN-FRONT-OF(A,D)
	IN-FRONT-OF(C,G)
	IN-FRONT-OF(J,C)
	IN-FRONT-OF(F,J)

	IN-FRONT-OF(K,E)
	IN-FRONT-OF(H,K);
	ON-STATION(G)
	ON-STATION(B)
	ON-STATION(E)
	...

Our solution proposes as seen in the table ZZZ, that the Stack is reorganized to focus on one railway at a time, allowing to solve the order of the wagons for it, and not having to reorganize it in future steps. In addition, the Predicates required to set up a railway are added so that they must be satisfied in the adequate order. The Predicate at the top will be ON-STATION and the IN-FRONT-OF predicates will follow in the order the wagons are in the railway starting from the Station.

Final State	Stack

|-G-C-J-F
|-B-D-A
|-E-K-H
Loc[]
	IN-FRONT-OF(F,J)
	IN-FRONT-OF(J,C)
	IN-FRONT-OF(C,G)
	ON-STATION(G)
	IN-FRONT-OF(A,D)
	IN-FRONT-OF(D,B)
	ON-STATION(B)
	IN-FRONT-OF(H,K)
	IN-FRONT-OF(K,E)
	ON-STATION(E)
	...

During this stage the algorithm attempts to stack the Goals in a manner where the Operator applied first satisfy automatically other Goals, that way we can guarantee consistency and ignore some of the lower priority Goals, e.g. If the set of Goals to be added to the Stack contains FREE(A) and IN-FRONT-OF(A,B), the system will order the Goals so that IN-FRONT-OF(A,B) is executed first, guaranteeing that FREE(A) will be true, that way the adding of this Predicate is complementary.

Final Load Stage:

Once the wagons are all in place, only remains to apply the Load operators to the relevant wagons that must be loaded or unloaded and the arrive to the station on the final state.

The final stage must end with the FREE_LOCOMOTIVE operator if needed.

Predicate groups

An analysis of the problem **State definition,** not the Operators definition, shows that the list of predicates can be classified in two categories:

- **Explicit**, includes the predicates that define the position of the wagons in a unique way and that if not included could cause ambiguities in the state representation. In this category we include:
    - ON-STATION
    - IN-FRONT-OF
    - TOWED
    - LOADED
    - EMPTY
    - n < MAX_RAILWAYS
- **Implicit**, refers to predicates that can be inferred from the rest of the state definition. The predicates in this category will be:
    - FREE
    - FREE-LOCOMOTIVE
    - USED_RAILWAYS

The implementation proposed benefits from this classification in two ways:

- When adding to the stack the predicates for the Final State, the Implicit Predicates, are added first (in fact, it would not have been required to add them at all), then the Explicit Predicates are added. In addition, the Explicit Predicates are also added in the order dictated by other heuristics used.
- When an Implicit Predicate is unstacked, if not satisfied, nothing is done to try to satisfy it as the Domain guarantees that when the rest of Predicates, either for the State or for the Operator, are satisfied this will be also satisfied in our implementation given the heuristics in place and the State definition. The Predicate is validated when the group it belongs to is unstacked. It is safe to assume this since the world is not dynamic and deterministic, otherwise the implicit predicates would have to be severely checked.

Domain Information

In order to provide the planner with reasoning and long term planning capabilities, some domain information has been included in the implementation to guide the choice of the Operators to satisfy the Predicates.

- ON-STATION(x): will force a railway to be emptied to place x, choosing the railway according to the heuristics described and will force the wagons preventing x to be towed to be towed themselves and moved to a railway that is not the railway that is being emptied to place x.
- IN-FRONT-OF(x, y): analogously to ON-STATION(x), it will force the wagons blocking y to be removed and as well as the wagons blocking x, move them to a different railway.

Those two heuristics are required to compensate for the lack of long term planning that linear planners have. The heuristic is still "legal" as the operator selection only uses the information of the current state.

If in this state:

```
|-G-(A)
|-B-(E)
|-D-C-(F)
Loc[]
```

the Predicate: ON-STATION(C) was found, the planner would choose railway 1 (where G is) and start moving the Wagons to railway 2 (Attach(A,E)), as moving them to railway 3 would be counterproductive to later move C. Once railway 1 is empty:

```
|-
|-B-(E)-(A)-G
|-D-C-(F)
Loc[]
```

the Planner would move F to railway 2, as moving it to railway 3 or 1 would be counterproductive too:

```
|-
|-B-(E)-(A)-G-(F)
|-D-C
Loc[]
```

Not adding this Domain Information causes the Planner to enter inefficiencies, even considering Partial Operator Instantiation, as the Planner does not retain memory about the Goals that made a certain Operator or Predicate being stacked so when solving partially instantiated operators it takes decisions without considering all the information required. The alternative would be to carry this information on the state definition but the "tuning" of the Linear Planner philosophy would be equivalent.

RESULTS

The system managed to complete successfully all the examples under which it was tested. After analysis the results of our approach were promising.

5 formal tests were undergone. Each of increasing complexity, also some of the special cases that should be considered in the task were tested in smaller and clearer examples. The first two tests are defined in the statement of the problem. All the others were designed to test how well does the Linear Goal Stack Planner performs under different situations.

The Resolution Plan shows the operators in inverse order as for debugging it was easier to have the last added operator in the first position.

Test 1:

Initial State	Final State
|-(A) |-B |-D-(C) Loc[]	|-(B) |-A |-(C)-(D) Loc[]

Test file: problem1.txt

Resolution plan: LOAD(B); UNLOAD(A); LOAD(D); COUPLE(A); ATTACH(A,B); DETACH(C,D); PARK(C); COUPLE(D); ATTACH(D,C); DETACH(A,B); PARK(A);

Steps: 11

Analysis: Since this was the first problem to be solved, the exercise was resolved by hand before the implementation. The manual resolution counted 9 steps, our first version achieved this number of steps as well. However when generalizing our heuristics to better solve more complex problems as well, the number of steps increased to 11. We kept this result because it allowed us to perform better on most cases while still maintaining a low number of steps in this example.

Test 2:

Initial State	Final State
|-B-(A) |-E-D-(C) |-(G)-F Loc[]	|-(A)-G |-B-(E) |-D-C-(F) Loc[]

Test file: problem2.txt

Resolution plan: UNLOAD(G); LOAD(E); DETACH(F,G); ATTACH(F,A); COUPLE(G); ATTACH(G,F); DETACH(C,D); PARK(C); UNLOAD(C); COUPLE(C); ATTACH(C,D); DETACH(G,F); ATTACH(G,C); DETACH(F,A); PARK(F); LOAD(F); COUPLE(F); ATTACH(F,A); DETACH(G,C); ATTACH(G,F); DETACH(C,D); ATTACH(C,G); DETACH(D,E); PARK(D); DETACH(C,G); ATTACH(C,D); DETACH(G,F); ATTACH(G,E); DETACH(F,A); ATTACH(F,C); DETACH(A,B); ATTACH(A,F); DETACH(G,E); ATTACH(G,A); COUPLE(E); ATTACH(E,B); DETACH(G,A); ATTACH(G,E); DETACH(A,F); PARK(A); DETACH(G,E); ATTACH(G,A);

Steps: 42

Analysis: This problem was also resolved manually before the implementation. In the manual version of the plan, this was comprised of 44 steps. The automatic resolution managed to be two steps shorter, so we believe that a very good solution was achieved in this problem. The difference between solving it in 50 steps (our previous best record) and in 42 steps lied in the tie breaking when selecting the shortest railway to disassemble.

Test: Consider adding a new railway

Test 3.1:

Initial State	Final State
|-(A)-B-(F) |-E-(D)-(C)-G |-(H)-I-J-(K) Loc[]	|-G-(C)-(J)-F |-(B)-D-(A)-I |-E-(K)-H Loc[]

Test file: problem3.1.txt

Resolution plan: UNLOAD(H); DETACH(F,B); ATTACH(F,G); DETACH(B,A); ATTACH(B,F); COUPLE(A); ATTACH(A,B); DETACH(K,J); ATTACH(K,A); DETACH(J,I); PARK(J); LOAD(J); COUPLE(J); ATTACH(J,I); DETACH(K,A); ATTACH(K,J); DETACH(A,B); ATTACH(A,K); DETACH(B,F); ATTACH(B,A);

DETACH(F,G); PARK(F); UNLOAD(F); COUPLE(F); ATTACH(F,B); DETACH(G,C); ATTACH(G,F); DETACH(C,D); ATTACH(C,G); DETACH(D,E); PARK(D); UNLOAD(D); COUPLE(D); ATTACH(D,E); DETACH(C,G); ATTACH(C,D); DETACH(G,F); ATTACH(G,C); DETACH(F,B); ATTACH(F,G); DETACH(B,A); PARK(B); DETACH(F,G); ATTACH(F,A); DETACH(G,C); ATTACH(G,F); DETACH(C,D); ATTACH(C,G); DETACH(D,E); ATTACH(D,B); DETACH(C,G); ATTACH(C,E); DETACH(G,F); ATTACH(G,C); DETACH(F,A); ATTACH(F,G); DETACH(A,K); ATTACH(A,D); DETACH(K,J); ATTACH(K,F); DETACH(J,I); ATTACH(J,K); DETACH(I,H); ATTACH(I,A); COUPLE(H); ATTACH(H,I); DETACH(J,K); ATTACH(J,H); DETACH(K,F); ATTACH(K,J); DETACH(F,G); ATTACH(F,K); DETACH(G,C); PARK(G); DETACH(C,E); ATTACH(C,G); DETACH(F,K); ATTACH(F,E); DETACH(K,J); ATTACH(K,F); DETACH(J,H); ATTACH(J,C); DETACH(K,F); ATTACH(K,H); DETACH(F,E); ATTACH(F,J); DETACH(K,H); ATTACH(K,E); DETACH(H,I); ATTACH(H,K); LOAD(B);

Steps: 91

Test 3.2:

Initial State	Final State
|-(A)-B-(F) |-E-(D)-(C)-G |-(H)-I-J-(K) | Loc[]	|-G-(C)-(J)-F |-(B)-D-(A)-I |-E-(K)-H | Loc[]

Test file: problem3.2.txt

Resolution plan: UNLOAD(H); DETACH(K,J); ATTACH(K,F); DETACH(J,I); PARK(J); LOAD(J); COUPLE(J); ATTACH(J,G); DETACH(K,F); ATTACH(K,J); DETACH(F,B); PARK(F); UNLOAD(F); COUPLE(F); ATTACH(F,B); DETACH(K,J); ATTACH(K,F); DETACH(J,G); ATTACH(J,K); DETACH(G,C); ATTACH(G,J); DETACH(C,D); ATTACH(C,G); DETACH(D,E); PARK(D); UNLOAD(D); COUPLE(D); ATTACH(D,E); DETACH(C,G); ATTACH(C,D); DETACH(G,J); ATTACH(G,C); DETACH(J,K); ATTACH(J,G); DETACH(K,F); ATTACH(K,J); DETACH(F,B); ATTACH(F,K); DETACH(B,A); PARK(B); DETACH(F,K); ATTACH(F,A); DETACH(K,J); ATTACH(K,F); DETACH(J,G); ATTACH(J,K); DETACH(G,C); ATTACH(G,J); DETACH(C,D); ATTACH(C,G); DETACH(D,E); ATTACH(D,B); DETACH(C,G); ATTACH(C,E); DETACH(G,J); ATTACH(G,C); DETACH(J,K); ATTACH(J,G); DETACH(K,F); ATTACH(K,J); DETACH(F,A); ATTACH(F,K); COUPLE(A); ATTACH(A,D); DETACH(I,H); ATTACH(I,A); DETACH(F,K); ATTACH(F,H); DETACH(K,J); ATTACH(K,F); DETACH(J,G); ATTACH(J,K); DETACH(G,C); PARK(G); DETACH(C,E); ATTACH(C,G); DETACH(J,K); ATTACH(J,C); DETACH(K,F); ATTACH(K,E); DETACH(F,H); ATTACH(F,J); COUPLE(H); ATTACH(H,K); LOAD(B);

Steps: 85

Analysis: The Test 3.1 and the Test 3.2 are the same problem with 11 letters and a medium level of complexity but with a different number of railways. In the Test 3.1, the problem is proposed with 3 railways and in the Test 3.2 the problem is with 4 railways. This test was proposed as a third exercise to solve, and to test if the plan was making good use of an extra railway. We have saved the plan 6 steps by adding an extra railway to the Plan. This happens because the worst case for accessing a wagon (having to move all the wagons in the same railway) has a lower expected value since the wagons will be splitted among more railways and the average length of wagons will be smaller.

Test 4: A new railway with High Entropy

Test 4.1:

Initial State	Final State
|-A-(B)-C |-D-E-(F)-G-(H) |-J-(K)-L-(M)-(N)-I Loc[]	|-C-G-I-N-(M)-(A) |-B-(F)-E-D |-(K)-(J)-H-(L) Loc[]

Test file: problem4.1.txt

Resolution plan: LOAD(J); LOAD(A); DETACH(C,B); ATTACH(C,I); DETACH(B,A); ATTACH(B,C); COUPLE(A); ATTACH(A,B); DETACH(H,G); PARK(H); UNLOAD(H); COUPLE(H); ATTACH(H,G); DETACH(A,B); ATTACH(A,H); DETACH(B,C); ATTACH(B,A); DETACH(C,I); ATTACH(C,B); DETACH(I,N); ATTACH(I,C); DETACH(N,M); PARK(N); UNLOAD(N); COUPLE(N); ATTACH(N,I); DETACH(M,L); ATTACH(M,N); DETACH(L,K); PARK(L); LOAD(L); COUPLE(L); ATTACH(L,K); DETACH(M,N); ATTACH(M,L); DETACH(N,I); ATTACH(N,M); DETACH(I,C); ATTACH(I,N); DETACH(C,B); PARK(C); DETACH(B,A); ATTACH(B,I); DETACH(A,H); ATTACH(A,B); DETACH(H,G); ATTACH(H,A); DETACH(G,F); ATTACH(G,C); DETACH(H,A); ATTACH(H,F); DETACH(A,B); ATTACH(A,H); DETACH(B,I); ATTACH(B,A); DETACH(I,N); ATTACH(I,G); DETACH(N,M); ATTACH(N,I); DETACH(M,L); ATTACH(M,N); DETACH(B,A); ATTACH(B,L); DETACH(A,H); ATTACH(A,M); DETACH(H,F); ATTACH(H,A); DETACH(F,E); ATTACH(F,H); DETACH(E,D); ATTACH(E,F); COUPLE(D); ATTACH(D,E); DETACH(B,L); ATTACH(B,D); DETACH(L,K); ATTACH(L,B); DETACH(K,J); PARK(K); COUPLE(J); ATTACH(J,K); DETACH(L,B); PARK(L); DETACH(B,D); ATTACH(B,L); DETACH(D,E); ATTACH(D,B); DETACH(E,F); ATTACH(E,D); DETACH(F,H); ATTACH(F,E); DETACH(H,A); ATTACH(H,J); DETACH(F,E); ATTACH(F,A); DETACH(E,D); ATTACH(E,F); DETACH(D,B); ATTACH(D,E); DETACH(B,L); ATTACH(B,D); COUPLE(L); ATTACH(L,H); DETACH(B,D); PARK(B); DETACH(D,E); ATTACH(D,L); DETACH(E,F); ATTACH(E,D); DETACH(F,A); ATTACH(F,B); DETACH(E,D); ATTACH(E,F); DETACH(D,L); ATTACH(D,E); UNLOAD(B);

Steps: 116

Test 4.2:

Initial State	Final State
|-A-(B)-C |-D-E-(F)-G |-(H)-I |-J-(K)-L-(M)-(N) Loc[]	|-C-G-I-N |-B-(F)-E-D |-(K)-(J)-H-(L) |-(M)-(A) Loc[]

Test file: problem4.2.txt

Resolution plan: LOAD(J); UNLOAD(H); LOAD(A); DETACH(I,H); ATTACH(I,C); COUPLE(H); ATTACH(H,I); DETACH(N,M); PARK(N); UNLOAD(N); COUPLE(N); ATTACH(N,H); DETACH(M,L); ATTACH(M,N); DETACH(L,K); PARK(L); LOAD(L); COUPLE(L); ATTACH(L,M); DETACH(K,J); PARK(K); COUPLE(J); ATTACH(J,K); DETACH(L,M); PARK(L); DETACH(M,N); ATTACH(M,G); DETACH(N,H); ATTACH(N,M); DETACH(H,I); ATTACH(H,J); COUPLE(L); ATTACH(L,H); DETACH(I,C); ATTACH(I,N); DETACH(C,B); ATTACH(C,I); DETACH(B,A); PARK(B); DETACH(C,I); ATTACH(C,A); DETACH(I,N); ATTACH(I,C); DETACH(N,M); ATTACH(N,I); DETACH(M,G); ATTACH(M,N); DETACH(G,F); ATTACH(G,M); DETACH(F,E); ATTACH(F,B); DETACH(E,D); ATTACH(E,F); COUPLE(D); ATTACH(D,E); DETACH(G,M); ATTACH(G,L); DETACH(M,N); ATTACH(M,G); DETACH(N,I); ATTACH(N,M); DETACH(I,C); ATTACH(I,N); DETACH(C,A); PARK(C); DETACH(I,N); ATTACH(I,A); DETACH(N,M); ATTACH(N,I); DETACH(M,G); ATTACH(M,N); DETACH(G,L); ATTACH(G,C); DETACH(M,N); ATTACH(M,L); DETACH(N,I); ATTACH(N,M); DETACH(I,A); ATTACH(I,G); DETACH(N,M); ATTACH(N,I); COUPLE(A); ATTACH(A,N); DETACH(M,L); PARK(M); DETACH(A,N); ATTACH(A,M); UNLOAD(B);

Steps: 88

Analysis: The Test 4 is meant to review our planner's management of increasing complexity. The amount of wagons is 14 now. 6 Operations of Load or Unload must be performed and no wagon is currently in the position it should be in at the moment. The test is resolved in 116 steps, comparing our results to the previous tests we can see that despite the increased complexity the amount of steps did not increase greatly. This is a good sign of the performance of our planner.

We also tested our planner's performance for this problem with 4 railways and a slightly different positioning to avoid starting with an empty railway as in the last example. As we can see again, there is a decrease of 28 steps in the number of steps to be performed until the Final State is achieved, we can appreciate even more how our planner handles effectively the fourth railway and uses it to improve its strategies.

TEST 5: Wagons a tutti pleni

Initial State	Final State
|-(A)-B-C-(D)-E-(F)-G-H |-I-J-K-(L)-M-(N)-O |-(P)-Q-(R)-S-T-(U)-V-W Loc[]	|-(H)-M-(K)-(W)-L-O-N |-B-D-U-(F)-(G)-A-C |-I-(Q)-T-(S)-(J)-(E)-P-R-V Loc[]

Test file: problem5.txt

Resolution Plan: UNLOAD(A); UNLOAD(P); DETACH(O,N); ATTACH(O,H); DETACH(N,M); ATTACH(N,O); DETACH(M,L); ATTACH(M,N); DETACH(L,K); ATTACH(L,M); DETACH(K,J); ATTACH(K,L); DETACH(J,I); ATTACH(J,K); COUPLE(I); ATTACH(I,J); DETACH(W,V); ATTACH(W,I); DETACH(V,U); ATTACH(V,W); DETACH(U,T); ATTACH(U,V); DETACH(T,S); ATTACH(T,U); DETACH(S,R); ATTACH(S,T); DETACH(R,Q); PARK(R); UNLOAD(R); COUPLE(R); ATTACH(R,Q); DETACH(S,T); PARK(S); LOAD(S); COUPLE(S); ATTACH(S,T); DETACH(R,Q); ATTACH(R,S); DETACH(Q,P); PARK(Q); LOAD(Q); COUPLE(Q); ATTACH(Q,P); DETACH(R,S); ATTACH(R,Q); DETACH(S,T); ATTACH(S,R); DETACH(T,U); ATTACH(T,S); DETACH(U,V); ATTACH(U,T); DETACH(V,W); ATTACH(V,U); DETACH(W,I); PARK(W); LOAD(W); COUPLE(W); ATTACH(W,I); DETACH(V,U); ATTACH(V,W); DETACH(U,T); PARK(U); UNLOAD(U); COUPLE(U); ATTACH(U,T); DETACH(V,W); ATTACH(V,U); DETACH(W,I); ATTACH(W,V); DETACH(I,J); ATTACH(I,W); DETACH(J,K); ATTACH(J,I); DETACH(K,L); PARK(K); LOAD(K); COUPLE(K); ATTACH(K,L); DETACH(J,I); PARK(J); LOAD(J); COUPLE(J); ATTACH(J,I); DETACH(K,L); ATTACH(K,J); DETACH(L,M); ATTACH(L,K); DETACH(M,N); ATTACH(M,L); DETACH(N,O); PARK(N); UNLOAD(N); COUPLE(N); ATTACH(N,O); DETACH(M,L); ATTACH(M,N); DETACH(L,K); PARK(L); UNLOAD(L); COUPLE(L); ATTACH(L,K); DETACH(M,N); ATTACH(M,L); DETACH(N,O); ATTACH(N,M); DETACH(O,H); ATTACH(O,N); DETACH(H,G); ATTACH(H,O); DETACH(G,F); PARK(G); LOAD(G); COUPLE(G); ATTACH(G,H); DETACH(F,E); ATTACH(F,G); DETACH(E,D); PARK(E); LOAD(E); COUPLE(E); ATTACH(E,F); DETACH(D,C); PARK(D); UNLOAD(D); COUPLE(D); ATTACH(D,C); DETACH(E,F); ATTACH(E,D); DETACH(F,G); ATTACH(F,E); DETACH(G,H); ATTACH(G,F); DETACH(H,O); ATTACH(H,G); DETACH(O,N); ATTACH(O,H); DETACH(N,M); ATTACH(N,O); DETACH(M,L); ATTACH(M,N); DETACH(L,K); ATTACH(L,M); DETACH(K,J); ATTACH(K,L); DETACH(J,I); ATTACH(J,K); DETACH(I,W); PARK(I); DETACH(W,V); ATTACH(W,J); DETACH(V,U); ATTACH(V,W); DETACH(U,T); ATTACH(U,V); DETACH(T,S); ATTACH(T,U); DETACH(S,R); ATTACH(S,T); DETACH(R,Q); ATTACH(R,S); DETACH(Q,P); ATTACH(Q,I); DETACH(R,S); ATTACH(R,P); DETACH(S,T); ATTACH(S,R); DETACH(T,U); ATTACH(T,Q); DETACH(S,R); ATTACH(S,T); DETACH(U,V); ATTACH(U,R); DETACH(V,W); ATTACH(V,U); DETACH(W,J); ATTACH(W,V); DETACH(J,K); ATTACH(J,S); DETACH(K,L); ATTACH(K,W); DETACH(L,M); ATTACH(L,K); DETACH(M,N); ATTACH(M,L); DETACH(N,O); ATTACH(N,M); DETACH(O,H); ATTACH(O,N); DETACH(H,G); ATTACH(H,O); DETACH(G,F); ATTACH(G,H); DETACH(F,E); ATTACH(F,G); DETACH(E,D); ATTACH(E,J); DETACH(F,G); ATTACH(F,D); DETACH(G,H); ATTACH(G,F); DETACH(H,O); ATTACH(H,G); DETACH(O,N); ATTACH(O,H); DETACH(N,M); ATTACH(N,O); DETACH(M,L); ATTACH(M,N); DETACH(L,K); ATTACH(L,M); DETACH(K,W); ATTACH(K,L); DETACH(W,V); ATTACH(W,K); DETACH(V,U); ATTACH(V,W); DETACH(U,R); ATTACH(U,V); DETACH(R,P); ATTACH(R,U); COUPLE(P); ATTACH(P,E); DETACH(R,U); ATTACH(R,P); DETACH(U,V); PARK(U); DETACH(V,W); ATTACH(V,R); COUPLE(U); ATTACH(U,V); DETACH(W,K); ATTACH(W,U); DETACH(K,L); ATTACH(K,W); DETACH(L,M);

ATTACH(L,K); DETACH(M,N); ATTACH(M,L); DETACH(N,O); ATTACH(N,M); DETACH(O,H); ATTACH(O,N); DETACH(H,G); ATTACH(H,O); DETACH(G,F); ATTACH(G,H); DETACH(F,D); ATTACH(F,G); DETACH(D,C); ATTACH(D,F); DETACH(C,B); ATTACH(C,D); DETACH(B,A); PARK(B); DETACH(C,D); ATTACH(C,A); DETACH(D,F); ATTACH(D,B); DETACH(F,G); ATTACH(F,C); DETACH(G,H); ATTACH(G,F); DETACH(H,O); ATTACH(H,G); DETACH(O,N); ATTACH(O,H); DETACH(N,M); ATTACH(N,O); DETACH(M,L); ATTACH(M,N); DETACH(L,K); ATTACH(L,M); DETACH(K,W); ATTACH(K,L); DETACH(W,U); ATTACH(W,K); DETACH(U,V); ATTACH(U,D); DETACH(W,K); ATTACH(W,V); DETACH(K,L); ATTACH(K,W); DETACH(L,M); ATTACH(L,K); DETACH(M,N); ATTACH(M,L); DETACH(N,O); ATTACH(N,M); DETACH(O,H); ATTACH(O,N); DETACH(H,G); ATTACH(H,O); DETACH(G,F); ATTACH(G,H); DETACH(F,C); ATTACH(F,U); DETACH(G,H); ATTACH(G,F); DETACH(C,A); ATTACH(C,H); COUPLE(A); ATTACH(A,G); DETACH(C,H); ATTACH(C,A); DETACH(H,O); PARK(H); DETACH(O,N); ATTACH(O,C); DETACH(N,M); ATTACH(N,O); DETACH(M,L); ATTACH(M,H); DETACH(L,K); ATTACH(L,N); DETACH(K,W); ATTACH(K,M); DETACH(W,V); ATTACH(W,K); DETACH(L,N); ATTACH(L,W); DETACH(N,O); ATTACH(N,V); DETACH(O,C); ATTACH(O,L); DETACH(N,V); ATTACH(N,O); LOAD(H);

Steps: 327

Analysis: This is a problem of high complexity. The station contains a total of 21 wagons. 15 of them must be loaded or unloaded, and only 3 of them satisfy ON-STATION at the Initial State or at the Final State. All the other 18 should be positioned to satisfy ON-STATION at an intermediate part of the plan. And the order is shuffled randomly amongst all the cards. Despite this and although the result is hard to measure without having other means of comparing, our belief is that the problems performs rather well, achieving a solution in 327 steps.

Special Test: Sussman Anomaly

Initial State	Final State
|-A-C |-B | Loc[]	|-C-B-A | | Loc[]

Test file: sussman.txt

Resolution Plan: DETACH(C,A); PARK(C); COUPLE(B); ATTACH(B,C); COUPLE(A); ATTACH(A,B);

Analysis: The Sussman Anomaly is a special case, where poor ordering of the goals might cause the planner to undo previously achieved goals. This was one of the situations we identified initially to be avoided. This special test was made to make sure that the case was been treated correctly. Analyzing the plan we can see that the order of the goals and railways selected allows us to avoid this special case, at least in given simple situations.

Special Test: Dismantle Choice

Initial State	Final State
|-G-(A) |-B-(E) |-D-C-(F) Loc[]	|-(A)-G |-B-(E) |-D-C-(F) Loc[]

Test file: dismantleChoice.txt

Resolution Plan: DETACH(A,G); ATTACH(A,E); COUPLE(G); ATTACH(G,F); DETACH(A,E); PARK(A); DETACH(G,F); ATTACH(G,A);

Analysis: This case was designed to test the intelligence of our planner when dismantling railways to achieve a goal. When our algorithm applies the property explained in the subsection Domain Information, It must select the best possible way to achieve its "long term" goals. This was designed to test this. In it the Initial State and the Final State are the same except for the order of the positions of A and G, which means that two out of three railways are satisfying their respective goals. Common sense says that the railway to be dismantled should be the one containing A and G (the only one not satisfying the State) so their order can be reversed.

In this test, the Goal to be achieved is ON-STATION(A), so the planner chooses to disassemble the first railway, the shortest and most disordered one, to correctly position these wagons.

Goal Stack	Current State
15|ON-STATION(A)←Current long term objective 14|IN-FRONT-OF(G,A) 13|ON-STATION(D)	|-G-(A) ← Selected railway to dismantle |-B-(E) |-D-C-(F) Loc[]

EXECUTING THE PLANNER

With this delivery will be attached the Eclipse project for the Planner. In it is the source code and binary files for executing the algorithm. The main class to run the system is found in the mai.par.Main. to select the test to run, the global variable file for the class can be changed to any of the files found in the “tests” folder.

There is also a .bat file, where the parameter -t will indicate which of the files found in the “tests” folder will be executed. If no parameter -t is given, it will execute the “./tests/problem1.txt” file set as default.

A step by step guide of the solution will be generated on a file of the same name as the one executed as parameter -t but the output will be given on the folder “log”. In it, the problem will be graphically described and solved step by step. Finally giving the Final State and the final plan.

CONCLUSIONS

The Linear Planner is not the best solution for the problem to be solved since solving it efficiently requires to tweak the algorithm significantly and, somehow, breaking the philosophy of Goal Stack Linear Planner.

The Linear Goal Stack Planner was completed successfully and tested thoroughly to find situations where its implementation could fail. So far the algorithm has remained robust and fast. As expected the plans developed are not necessary the global optimal, but given the constraints that apply to linear planning the solutions found are efficient.

One of the things that helped to develop a robust strategy set for the algorithm was manually solving the problems. This way from the experience could be extracted a group of “Expert” heuristics and rules that could lead the algorithm down the correct path. Also the Divide and Conquer approach taken was a good contribution to make the algorithm have clear Goals that would not fall for the Sussman Anomaly and other undesired cases, given that the linear planner is not exhaustive in the exploration strategy and that it should only reason based on the information available on the first position of the Stack and the current state.

Among the choices we made, we found that the Operators Groupings made the system much more robust and intelligent. The Operators were simplified, to express for example the Take and Drop operator together. This sacrifices flexibility for speed and consistency. However this solution is good as long as it remains the number of locomotives as one. In the case of adding another locomotive, it would be interesting to split this operators and look for other solutions with this new parameter. However this escapes the scope of this work.

Another possible strategy would be to select the rails to dismantle or to arrange first, according to the chaos found in them. This is the number of wagons that are correctly positioned in them. However we can think of various cases where this would not outperform our chosen solutions. This type of heuristics would have to be investigated thoroughly in practice before replacing our current good selections.

Overall the results are promising, and the railway would work efficiently and efficaciously. Further tests could be performed to find situations not foreseen on which our planner could not converge to a plan, or the given plan would be far from optimal.

REFERENCES

[1] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, p. 414, ISBN 0-13-790395-2