Example report: Controlling a vacuum cleaner without location sensors

Jan Frederik Schaefer

November 29, 2022

Abstract

This report explores the challenge of controlling a robot with incomplete information. More concretely, we will explore how to create efficient cleaning plans for a vacuum cleaner robot without knowing its initial position. As a baseline we will use randomly generated plans. It turns out that with a few very simple strategies, we can obtain much shorter cleaning plans.

I decided against a table of contents because the report is so short.

1 Introduction

Agents perceive the world via sensors and change it by performing actions. The goal of artificial intelligence is to compute reasonable actions for an agent given its perception of the world. A key challenge is that sensors usually provide incomplete information. The reasons are varied, ranging from technical limitations to cost considerations. Nevertheless, agents can perform surprisingly well with very limited sensor information. In this report we will explore that using the example of a vacuum cleaner robot in a grid world that has no location sensors.

Marking the research question and contribution that explicitly is not necessary, but makes it easier to read (and write).

Research question How can you control a vacuum cleaner robot without location information?

Contribution There are two key contributions in this report:

1. an efficient way to represent the cleaning state,
2. an evaluation of several simple strategies to find short cleaning paths.

See also comment 1 in the appendix.

Overview First, the problem is described in more detail in Section 2. Section 3 proposes an efficient representation of the cleaning state, which is evaluated in Section 4. Section 5 discusses different strategies to find short plans and Section 6 concludes the report.

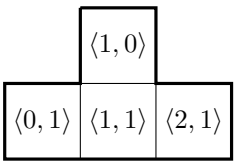

Figure 1: Example apartment with coordinates (the origin is in the top-left corner).

2 Problem description

We represent an apartment as a set of connected cells in a rectilinear grid. Each cell is described by its integer coordinates. For example, the apartment in Figure 1 would be $R = \{\langle 1, 0\rangle, \langle 0, 1\rangle, \langle 1, 1\rangle, \langle 2, 1\rangle\}$.

Our goal is to find a cleaning plan that would clean the entire apartment. A cleaning plan is a sequence $[a_1, \ldots, a_n]$, where each a_i represents an attempt to move the vacuum cleaner. There are 4 possible motions:

- $N := \langle 0, -1\rangle$ moves the vacuum cleaner one square *north*,
- $E := \langle 1, 0\rangle$ moves it *east*,
- $S := \langle 0, 1\rangle$ moves it *south*,
- $W := \langle -1, 0\rangle$ moves it *west*.

If a motion a_i would move the vacuum cleaner outside the apartment, it will instead remain in its current location.

For example, the plan $p := [E, N, S, E]$ would clean the example apartment R (Figure 1) if we start in cell $\langle 0, 1\rangle$, but it would miss some squares if we start in a different square. Since the starting position is unknown, we need a plan that would work for any starting position. The sequence $[S, W, W]$ ensures that the vacuum cleaner is in $\langle 0, 1\rangle$ (recall that motions are ignored if they would move the cleaner outside R). Appending p, we would get the plan $p' := [S, W, W, E, N, S, E]$, which is guaranteed to clean R completely for any starting position.

It would have been possible to just say *"A cleaning path is a sequence of motion attempts"*, but this way we've also introduced a notation for it.

Inconsistent terminology: Do we clean *squares* or *cells*? Using two different words for the same thing is confusing and should be avoided in technical language.

See also comments 2, 3, and 4 in the appendix.

3 Representing the Cleaning State

The plan finding algorithms discussed in this report incrementally extend a plan until everything gets cleaned. It is very helpful to have a compact representation of the "cleaning state" after executing an incomplete plan.

Let us re-use the example apartment from Figure 1 and use the example cleaning plan $p = [E, N, S]$. If we assume that the starting position is $\langle 1, 1\rangle$, then we could represent the cleaning state after performing p simply by the set of cleaned squares ($\{\langle 1, 1\rangle, \langle 2, 1\rangle\}$) and the current position of the vacuum cleaner ($\langle 2, 1\rangle$).

Since we do not know the starting position, we could store that information for every possible starting position. For the example, we could use the following table to represent the cleaning state after performing $[E, N, S]$:

It is often useful to explain what you want to do (and why) before going into the details.

Re-using examples (or using a running example) avoids the overhead of introducing many different examples.

Starting position	cleaned squares	current position
$\langle 0,1\rangle$	$\{\langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$	$\langle 1,1\rangle$
$\langle 1,0\rangle$	$\{\langle 1,0\rangle, \langle 1,1\rangle\}$	$\langle 1,1\rangle$
$\langle 1,1\rangle$	$\{\langle 1,1\rangle, \langle 2,1\rangle\}$	$\langle 2,1\rangle$
$\langle 2,1\rangle$	$\{\langle 2,1\rangle\}$	$\langle 2,1\rangle$

We will refer to this representation as the **naive representation**. The disadvantage of the naive representation is that it is not very efficient. Alternatively, we can store for every possible current position C, what squares must have been cleaned to get there, which we will call the **trail of** C:

Writing is a lot easier if you name things (we use boldface to mark newly defined words).

Current position C	cleaned squares (trail of C) after $[E,N,S]$
$\langle 2,1\rangle$	$\{\langle 2,1\rangle\}$
$\langle 1,1\rangle$	$\{\langle 1,0\rangle, \langle 1,1\rangle\}$

We will call this the **trail approach**. The advantage is that trails tend to get merged, which results in a much smaller representation. In this case, only the squares $\langle 2,1\rangle$ and $\langle 1,1\rangle$ are reachable. Note that only $\langle 2,1\rangle$ is in the trail of $\langle 2,1\rangle$ after the plan $[E,N,S]$. Depending on the starting position, $\langle 1,1\rangle$ might have been cleaned as well, but it is not guaranteed.
Section 4 compares the efficiency of the two representations for a larger apartment.

Here it would have been possible to give an algorithm for updating the trail representation. In this case I assumed that it is easy enough to come up with an algorithm.

4 Comparing the State Representations

To compare the naive representation and the trail representation (Section 3), we use a square apartment of size 10×10 and a randomly generated sequence of motions. We incrementally update each representation with the next motion until everything is cleaned. Figure 2 shows the total number of squares stored in the representations after each update. As expected, the number of squares stored for the naive representations grows monotonically. The size of the trail representation, however, remains much smaller because most trails get merged early on.

See comment 5 in the appendix.

Incidentally, computing the state after 1000 motions took approximately 810 ms with the naive approach and 18 ms with the trail approach.

5 Finding Plans

The easiest way to find a plan is to start with an empty plan and to append randomly chosen motions until everything gets cleaned. We will take a look at three strategies to find shorter plans:

1. **Repeat strategy:** Repeat the algorithm 5 times. Since the algorithm is randomized, we can get different results and pick the best one.
2. **Greedy strategy:** Select the motion that extends most trails. If there are multiple candidates, pick randomly.
3. **No-redundancy strategy:** If a sequence of motions leads you to a cleaning state that you have already been in, you can simply eliminate that sequence. For example, the plan $[E,W,E,W,N]$

It should be justified why the number 5 was chosen. See also comment 6 in the appendix.

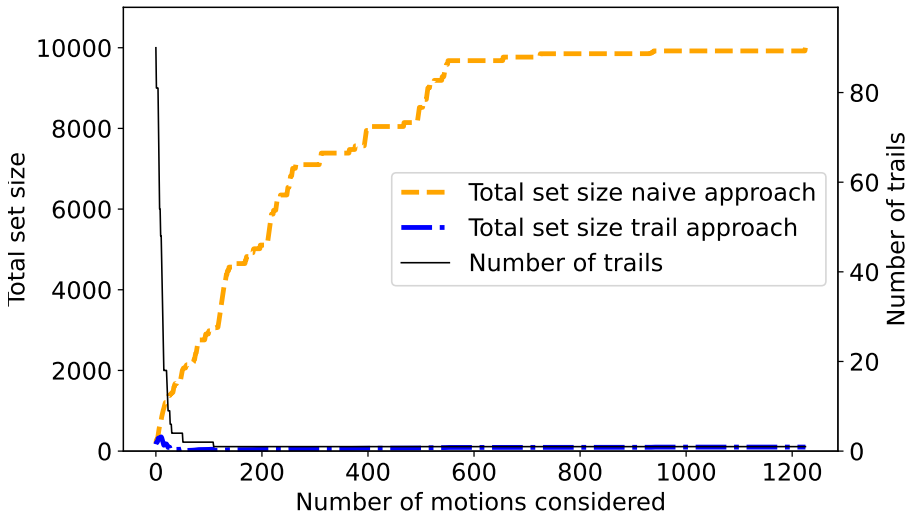

Figure 2: Comparison of the naive approach and the trail approach.

Strategies	Total plan length
-	19707
Repeat	10388
Greedy	8219
Repeat, Greedy	2976
No-redundancy	2947
Repeat, No-redundancy	2432
No-redundancy, Greedy	2072
Repeat, No-redundancy, Greedy	1656

Figure 3: Comparison of different strategy combinations.

can be shortened to $[E, W, N]$ because we reach the same state after $[E, W]$ and after $[E, W, E, W]$.

These strategies can be arbitrarily combined, which gives us $2^3 = 8$ different algorithms. For evaluation, I ran the algorithm over the problem files 80 – 99 in [1] and added up the lengths of the found plans. Figure 3 shows the results. It appears that each of the strategies significantly reduces the plan lengths with a combination of all three strategies leading to the shortest plans.

It would also be interesting here to compare the run times.

6 Conclusion

In this report we have explored how to make a cleaning plan for a vacuum cleaner on a rectilinear grid without knowing the starting position. A key challenge was to find a suitable representation of the cleaning state. It turns out that the trail representation described in Section 3) is much more efficient than a more naive representation. To find the actual cleaning plan, we generated a random plan and implemented three different strategies for shortening the plans: i) running the algorithm multiple times and picking the best result, ii) using a greedy approach, iii) removing redundant motions. In practice, combining all three strategies

resulted in the shortest plans.

References

[1] Jan Frederik Schaefer. *Assignment 0*. URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2223/a0-clean-the-wumpus-cave/assignment` (visited on 11/28/2022).

A Extra comments

1. If we write the contribution explicitly, the grader/reviewer does not have to guess. If the reviewer thinks we want to do X, but we actually wanted to do Y, we might get criticized for not solving X well.

2. The format of problem files is not discussed because it led to no interesting challenges.

3. The problem description only mentions plan finding and not plan checking.

4. This report does not have any preliminaries (that is basically by design for the warm-up problem). Your report probably will have some preliminaries.

5. Section 4 is basically an evaluation of the trail approach. This "violates" the standard document structure, where you put the evaluation to the very end. In this case, though, I thought that this order makes more sense. You should always structure your document in the way that conveys your ideas best. The standard document structure often makes sense (also because readers expect it), but sometimes a different structure is better.

6. Generally, we have to justify our design decisions. In this case (Section 5), it is unclear why the number 5 was chosen. But some decisions in the setup are not interesting, e.g. that the programming language Python was chosen or that apartments were represented as a list of lists. In fact, these design decisions were not even mentioned.