SENSOR SELECTION FOR BEHAVIOR VALIDATION OF MULTIPLE AGENTS

An Undergraduate Research Scholars Thesis

by

PATRICK ZHONG

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Dylan Shell

May 2022

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Patrick Zhong, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Sensor Selection for Behavior Validation of Multiple Agents

Patrick Zhong
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. Dylan Shell
Department of Computer Science and Engineering
Texas A&M University

Given a pre-declared itinerary of potential activities and sites for sensor placement within an environment, sensor selection involves choosing a set of sensors which can determine whether what actually occurs matches the supplied itinerary. This problem is encountered when, subject to some budget, one instruments a facility in order to ensure that the agents within behave as expected (e.g., a laboratory where the robots operating inside should follow some policy). It also applies to settings that range from surveillance and security to the design of smart spaces. We tackle a variant of the sensor selection problem where multiple agents share the same environment, which introduces some modeling subtleties, including those arising from interactions. Specifically, the multi-agent validation problem may require more than merely the union of sensors necessary for individual agents owing to aliasing: different agents may trigger sensors without those sensors necessarily being able to distinguish who was the cause. Also, the treatment of time and modeling of interleaving becomes important in providing joint itineraries, especially when combining itineraries of individuals. Since the underlying problem is NP-hard, when multiple agents are considered, another of the issues is the natural increase in size of problem instances. This paper re-formulates sensor selection as a SAT problem and introduces a graph trimming technique based on a reachability analysis. Treating the problem as a question of satisfiability is especially

apt when the primary interest is in determining whether the sensors that one has available (or are within some budget to purchase) have some arrangement that suffices to validate the itinerary of interest. It also facilitates use of fast, state-of-the-art solvers. Taken together, these modifications yield significant speed-up over the previous method, as we detail in our empirical results based on simple 2-agent case studies.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Dylan Shell, for his guidance, support, and assistance throughout the course of this research.

Thanks also go to my friends and colleagues, the department faculty and staff, and the ASTRO Lab for making my time at Texas A&M University a great and valuable experience.

The present thesis is a slightly extended treatment of the manuscript [1] currently under submission to the IEEE/RSJ International Conference on Intelligent Robots and Systems.

Funding Sources

Undergraduate research was supported by the College of Engineering at Texas A&M University. No funding was sourced for this research thesis.

1. INTRODUCTION

An important and basic capability for intelligent systems that operate in the real world is spatial awareness. For a robot, one usually wishes to know where the robot is physically and what it is doing (perhaps distinct from what it believes [2]). Or some other cyber-physical system may wish to answer those same questions about people or vehicles moving about the environment. One general approach is to fill the environment with large numbers of sensors, often quite capable devices, in order to obtain coverage [3] and then process their output streams with a variety of techniques to identify, track, and classify individuals and their behavior [4]. Apart from the financial and computational costs, this approach may capture all sorts of information that is irrelevant, or potentially even private or sensitive. For instance, when considering the design of smart spaces [5], like homes and offices equipped with IoT-based devices for activity monitoring [6], collecting more information than necessary is often quite undesirable.

Building upon the model in [7], the approach we consider is to declare up-front what behaviors the agents in the environment will engage in. Then, using this itinerary and the structure of the environment, to select sensors so that they can detect divergence from this specification. The requirement is essentially that the chosen sensors have adequate distinguishing power to discern, from a sequence of sensor readings, if any deviation from the itinerary has occurred. Whether any subset of sensors (including all of them) has sufficient discriminating power depends on the environment and the specific itinerary. And if it is feasible to sense deviations from the planned itinerary, one might also ask how many sensors are needed.

Couching the problem in terms of a known itinerary is useful because, in practice, the result will often be a relatively sparse set—the sensors need only answer the question: do the observations fit the itinerary or not? Furthermore, a pre-declared itinerary is especially useful for applications involving robots. For instance, a robot in a manufacturing facility may share an environment with human workers, but only be permitted to act when certain precise conditions are met (e.g., such

as all the workers are moving between particular subregions of the environment in some specific order). Those conditions become the itinerary and then we select sensors that detect when the humans are behaving differently from expected. Those sensors can tell the robot when it must stop its activity. Or, for a different instance, in an academic robotics laboratory where an experiment is about to be carried out, we might derive the itinerary directly from a policy describing how a team of robots is supposed to move. We then select peripheral sensors, operating as an exterior sub-system independent from the robots themselves, which serves as a safeguard by monitoring and reporting if something goes awry during the experiment.

Rahmani, Shell, and O'Kane [7] provide, in addition to the basic model, an integer linear programming (ILP) method for determining the minimum sensor selection (i.e., optimal sensor count and placement). In this paper, we developed a new and faster SAT formulation of the same underlying set of constraints which determine whether a sensor selection of a given size exists. The SAT formulation can be used in a stepping algorithm to determine the minimum sensor selection, and is able to solve larger problem instances than the ILP implementation; we also believe that there are circumstances where the question is inherently one of determining sufficiency (e.g., when considering the sensors already in one's possession).

As emphasized in the examples two paragraphs back, scenarios of practical interest may involve multiple people or robots. The presence of multiple agents can affect the functionality of sensors. For example, a basic occupancy sensor would not register any changes if an agent entered the room while another agent was already present. Additionally, multi-agent scenarios allow for more complex itineraries. This complexity can manifest as a rapid increase in the size of problem instances. The improved efficiency of our SAT-based approach helps in this regard, but even more impactful is a new trimming optimization which pares away unimportant states. Aside from this, in multiple-agent settings, complexity also increases in terms of specification subtlety. Whereas the original single-agent implementation has a treatment of time that is straightforward, multiple-agent scenarios require care in treating assumptions about the progression of time: for instance, whether synchronized/lock-step motion is being expressed in the itinerary or not. Additionally, validation

of multiple agents with each following their own preset itinerary is weaker than multiple agents with dependent itineraries. An example of the latter would be one agent tailing another, always remaining one step behind.

The optimizations that we introduce to speed up the solution method are necessary for it to be feasible to solve cases involving multiple agents in reasonable time, but another important contribution of the paper is in disentangling the component strands that make up the multiple agent case.

To communicate these aspects, the paper proceeds as follows: After describing relations to work within the literature next, Sections 3.1 and 3.2 will give the formal definition of the problem in single and multiple agent cases, respectively. Next, Sections 3.3 and 3.4 give a detailed description of the SAT formulation and pre-processing method that prunes surplus fragments of the problem. Section 4 reports our empirical findings, before the final section, which concludes.

2. RELATED WORK

As already mentioned, our work employs the theoretical model of [7] and, accordingly, we inherit the positive features of that formalization: it is rich for the reason that sensors can be grouped, so that a single selection can impact more than one location. Furthermore, the approach allows one to consider edge-oriented transitions rather than purely state-based properties, which distinguishes it from the (quite general) framework of [8]. For details on the relationship to the research on sensor activation, diagnosability, and fault detection problems, the reader is referred to [7, Section II] and references therein. Of course, we also inherit negative aspects of the formulation: the NP-hardness. Significant work was needed for it to be feasible to treat the case with 2 agents; the results we report are only for pairs as scaling-up further remains challenging.

The authors of that earlier work emphasize that their treatment of an itinerary (as a regular language of transitions) affords a degree of flexibility absent from similarly motivated prior work in the literature (e.g., validation for security and safety applications). For instance, comparing with [9], it is natural to need additional flexibility in specifying a future itinerary because making claims about the future is harder than making ones about the past. As our focus is on multiple agents, this flexibility becomes still more crucial: there are additional degrees-of-freedom, including in the interleaving of and dependencies between agents. Subsequent sections will draw out the distinction between joint itineraries and ones composed from individual itineraries of the agents.

We also note that there are examples of queries with sparse sensors (like the beams that are included here) where one may be able to answer a certain question about multiple agents despite the sensing being too weak to determine properties about individual agents. A concrete example is the set-up of [10, pg. 12], with three beam sensors in a circular environment, that can determine whether two agents are *together* or *apart* without actually knowing where they are.

Finally, sensor selection can be understood as a specialized type of robot design. In this paper, we are designing an exterior sensing system to complement the robot operating within the

environment. It constitutes a type of infrastructure for safety or sensing. This point of view is actually quite common in laboratory settings: one need think only of the prevalence of motion capture systems, or field experiments employing a differential GPS system. There is substantial existing work on minimizing resources for robot design (see, for instance [11]). In particular, selecting the fewest actuators [12] appears dual to this sensing-oriented problem.

3. METHODS

3.1 Basic Problem Definition

We start by giving some needed definitions; these are cut-down versions of the full, more formal definitions in [7].

First, we must model the environment. We consider planar environments which are segmented into regions where movement between regions has the potential to trigger a sensor response, had some sensor been chosen to be deployed there. The phrase 'sensor response' is used because we treat both sensors that detect a physical crossing and those which measure occupancy. The former might be a beam sensor guarding a doorway, whereas the latter could be a motion sensor or camera. The 'response' is either a beam being broken or a change in occupancy, as agents are detected entering or leaving the region. These primitive sensor types serve as an abstraction and may, of course, be implemented using more sophisticated devices.

The model treats the environment as a directed multigraph, known as the *world graph*, and represents its spatial structure by encoding regions with vertices and placing (potentially multiple) directed edges between them for transitions from one region to the next. Additionally, we have a finite set of *potential sensors* $S = \{s_1, s_2, \ldots, s_k\}$. For each sensor we associate a set of *events* that each sensor can produce: sensor s_k might give any of the readings in Y_{s_k}. (For $s_i \neq s_j$, every Y_{s_i} and Y_{s_j} are assumed to be disjoint.) Each edge in the world graph will bear a collection of events; for a single agent scenario (which is the one we begin with) these will be elements from $Y_{s_1} \cup Y_{s_2} \cup \cdots \cup Y_{s_k}$.

Figure 3.1(a) provides a very basic example: an environment consisting of six rooms arranged in the form of a 2×3 grid. The black markings represent walls. Some of the adjoining doorways (a total 5 in this figure) have been identified as candidate locations for beam sensors. These are marked (and labeled 'bh_{ij}' or 'bv_{ij}') in red. Additionally, some of the rooms may host an occupancy sensor, these are marked in blue (and labeled 'o_{ij}', numbering 4 in total here). In Figure 3.1(b), below the map, a visualization of the world graph is provided: edges are labeled

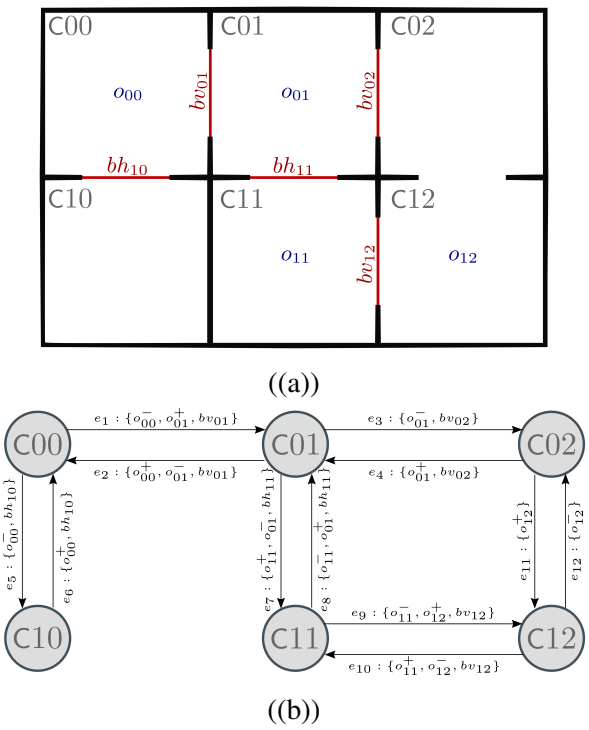


Figure 3.1: An example environment with 6 rooms distributed on a 2×3 grid and its world graph. The rooms and sensors are labeled according to row and column. Occupancy sensors are labeled 'o_{ij}' and beam sensors are labeled 'bh_{ij}' or 'bv_{ij}' for horizontal and vertical. (The mnemonic for C is 'cell'.)

to give them names (for example traversal from C00 to C01 is along the edge with name e_1) and they bear sets describing events that would generated if an agent were to move across an edge, and the associated sensor installed. For instance the event 'bv_{01}' on e_1 indicates that, should a vertical beam sensor be placed between C00 and C01, then it would have been triggered if an agent moved from the former region to the latter. Unlike beam sensors, the occupancy sensors have two events associated with them. These encode changes. Where occupancy is sensed in a previously unoccupied region, the event is marked with a '$+$'; a change from detection to absence of any, the reverse, has a '$-$' instead. Edge e_1 has, in addition to 'bv_{01}', the events 'o_{00}^-' and 'o_{10}^+' showing that those respective events would be generated if occupancy sensor o_{00} was deployed in region C00, and the same for o_{01} in C01.

Agents will move on some continuous paths that do not pass through any walls. Before any sensors are selected, in an environment with a single agent, that agent will describe how it intends

to move: for instance it might claim that it will move along some particular route. The claim forms an *itinerary*, which is modeled as a deterministic finite automaton (DFA) [13] that has world graph edges as its alphabet. The automaton representation allows the itinerary to incorporate not only a single route but multiple if there are choices which the agent will decide as it actually moves. It is also possible for the claim to have infinite paths, such as those involving loops. Continuing the example from Figure 3.1 above, an agent starting in C00 that moves east and then reverses and heads west back to the starting point again, would have an itinerary given as an automaton which accepts exactly two strings: e_1e_2 and $e_1e_3e_4e_2$ corresponding to the cases where it moves only one room over and back, or two rooms over before heading back, respectively.

The key idea for sensor selection is that some subset of the available sensors S may provide enough distinguishing power to tell, simply via sequences of triggered events, whether the generating motion in the environment matches the itinerary or does not. If so, it is said to certify:

Definition 1. [Certifying Sensor Selection [7]] Letting $M \subseteq S$ be a subset of sensors, we say M *certifies* itinerary $\mathcal{I}$ (whose language is $L_{\mathcal{I}}$) on world graph $\mathcal{G}$ if there exist no $r \in L_{\mathcal{I}} \cap \mathrm{Walks}(\mathcal{G})$ and $t \in \mathrm{Walks}(\mathcal{G}) \setminus L_{\mathcal{I}}$ such that r and t produce identical sensor event sequences, when only the sensors in M produce events.

The notation $\mathrm{Walks}(\mathcal{G})$ above denotes the set of all finite paths on world graph $\mathcal{G}$. The intuition is that M is a certifying sensor selection for $\mathcal{I}$ when, using only the events generated by the sensors in M, the system is able to discriminate whether r fits the claimed itinerary or not. This leads naturally to the basic question:

Minimal Sensor Selection to Validate an Itinerary [7]

Input: A world graph $\mathcal{G}$, an itinerary DFA $\mathcal{I}$, and $k \in \mathbb{N}$.

Output: YES if there exists a certifying sensor selection $M \subseteq S$ such that $|M| \leq k$; NO otherwise.

We will write MSSVI as an abbreviation for this problem.

It is helpful to understand how a certifying selection is actually used. Once the certifying sensor set M has been found, one can construct an DFA which has the associated events from

all the elements in M on its edges. Then the events that are generated by activities in the world are fed into this DFA in a streaming fashion. It has four modes of operation: (I) it may find, in processing a sequence, that the prefix of the sequence already violates the itinerary DFA $\mathcal{I}$ and this is determined as soon as this infringement is conclusive; (II) it may be processing a sequence that has not yet ended, that there is not yet an indication that the itinerary has been violated thus far; (III) reaching the end of the sequence, it arrives in an accepting state, and one can declare that it meets the itinerary; (IV) reaching the end, it does not complete all the steps demanded of the itinerary, so does not reach an accepting state, violating the itinerary because it is only a prefix of the stated behavior. The prior work does not belabour these cases and they are not needed in such detail merely for formulating MSSVI itself, but they are quite helpful for understanding the general scheme employed.

3.2 Treatment of Multiple Agents

The starting point for sensor selection with multiple agents is to treat it via the preceding MSSVI problem. The basic idea is to represent the multi-agent problem using joint world graphs and joint itineraries that, while structurally no different than any other world graph or itinerary, encode information that relates the multi-agent part of the problem. Each vertex, state, edge, or transition in the joint world graph or joint itinerary represents the position or movement of multiple agents. Once in this form, the problem can be solved in the same way, in principle, using a world graph and itinerary.

3.2.1 Models of Time and Interaction

The single-agent problem does not need to consider time very explicitly. Indeed, much is hidden behind the word "produce" in Definition 1. The agent moves and its actions trigger signal changes that yield a sequence of event symbols. With certifying sensors, the tracing of those symbols permits detection of some violation when it occurs. Thus, the processing is event-driven, and the agent is the prime mover.

When there are multiple agents we must ask whether a symbol might be generated by any of the agents, or whether we are modeling a scenario where (perhaps, extra domain knowledge

indicates that) motions will follow a more tight-knit, synchronous pattern. We will consider two possible models for time-stepping:

Lockstep is the direct analog of the single-agent case: we assume there is no loitering allowed by any of the agents, they must move together at the same time.

Independent time-stepping allows for agents to loiter if at least one other agent is moving.

Note that these are assumptions on motions, not the event sequences—some motions may not generate events: e.g., in the example environment in Figure 3.1(a), some traversal from C02 to C12, when occupancy sensor o_{01} was not part of the selection. Thus, even if all the agents follow such a motion pattern that is strictly synchronous, one cannot assume that they will always trigger events simultaneously.

Of the two models, lockstep is the more restrictive (representing a sort of universal quantification). It is easier to solve, but also less representative of many actual scenarios. As will discussed in some detail in the subsection that follows, the joint world graph in this model is simply obtained by a product of world graphs. The case of independent motion is more permissive (representing an existential quantifier) and the joint graph here has extra edges added to express the additional degrees of freedom given to the agents. Also, as one might possibly expect, these transitions can lead to longer solution times.

3.2.2 The Joint World Graph and Itinerary

We provide some additional information on how to formulate the multi-agent itinerary validation problem as a special, large single-agent problem by creating a single joint world graph and joint itinerary that represents all the agents.

3.2.2.1 Joint World Graph

The joint world graph is created by tensor producting a series of world graphs, one for each agent. Each vertex represents the combined state of all the agents, and each edge represents a set of actions, one per agent. Also, the initial vertex is the combination of each agent's starting vertex, thus allowing for different starting points. Figure 3.2 provides a small sample of a joint graph

based in Figure 3.1, but now for two agents. The pair of labels $\binom{\mathsf{C00}}{\mathsf{C01}}$ indicate one agent in each of C00 and C01, respectively.

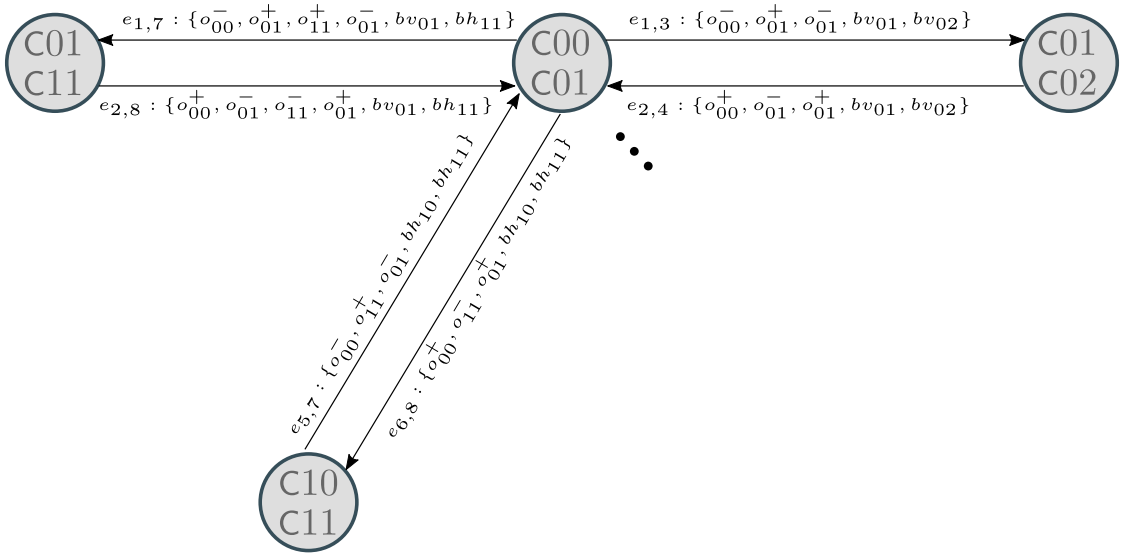


Figure 3.2: A tiny portion of the joint world graph for two agents in the 2×3 grid environment of Figure 3.1.

By default the labels of each edge, which detail the sensor events emitted when the edge is taken, is the set union of the labels of the edges combined to make it. This behavior may be modified in order to capture the various subtleties in how multiple agents can affect sensors. For example, the '$+$' event for an occupancy sensor can be removed from all the edges with agents entering the room if an agent is already in the room, and thus preventing the occupancy sensor from triggering multiple times if it was never released. Also, we can generalize to more powerful sensors. For instance, if the sensor were not an occupancy sensor but instead was a counter, this would be implemented as follows: rather than the '$+$' and '$-$' pair of superscripts, we might use '0', '1', '2', ..., 'n' for the n agent case. It is also possible to extend, in a simple way via a Cartesian product, the set of events for sensors that are capable of telling the individual agents apart. For instance, with 2 agents, if the occupancy sensor is able to distinguish them, we have six superscripts of the form '$\langle x, y \rangle$' where $x, y \in \{+, -, \emptyset\}$, encoding changes in detected occupancy

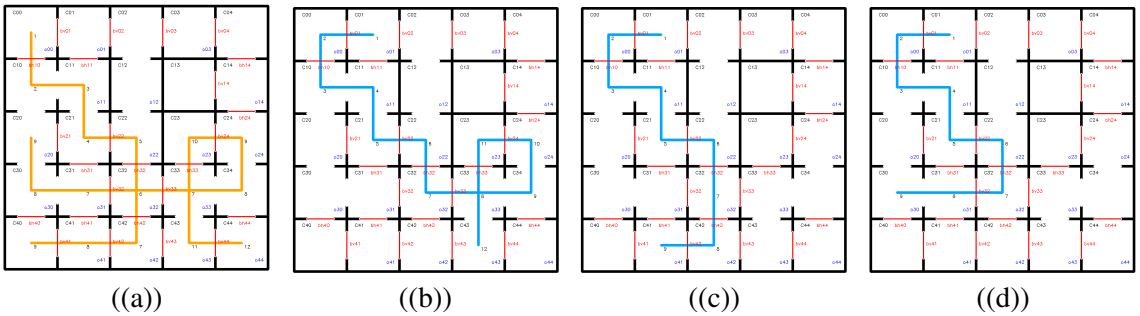
for first and second agent, respectively.

Without adding any additional edges, this graph product models the lockstep assumption. For independent motion, we add additional edges to the joint world graph that allow each agent to loiter while the other agent is moving. Like lockstep, independent time-stepping still only steps when there are sensor events emitted: situations where all the agents are loitering will not generate any event so are not included.

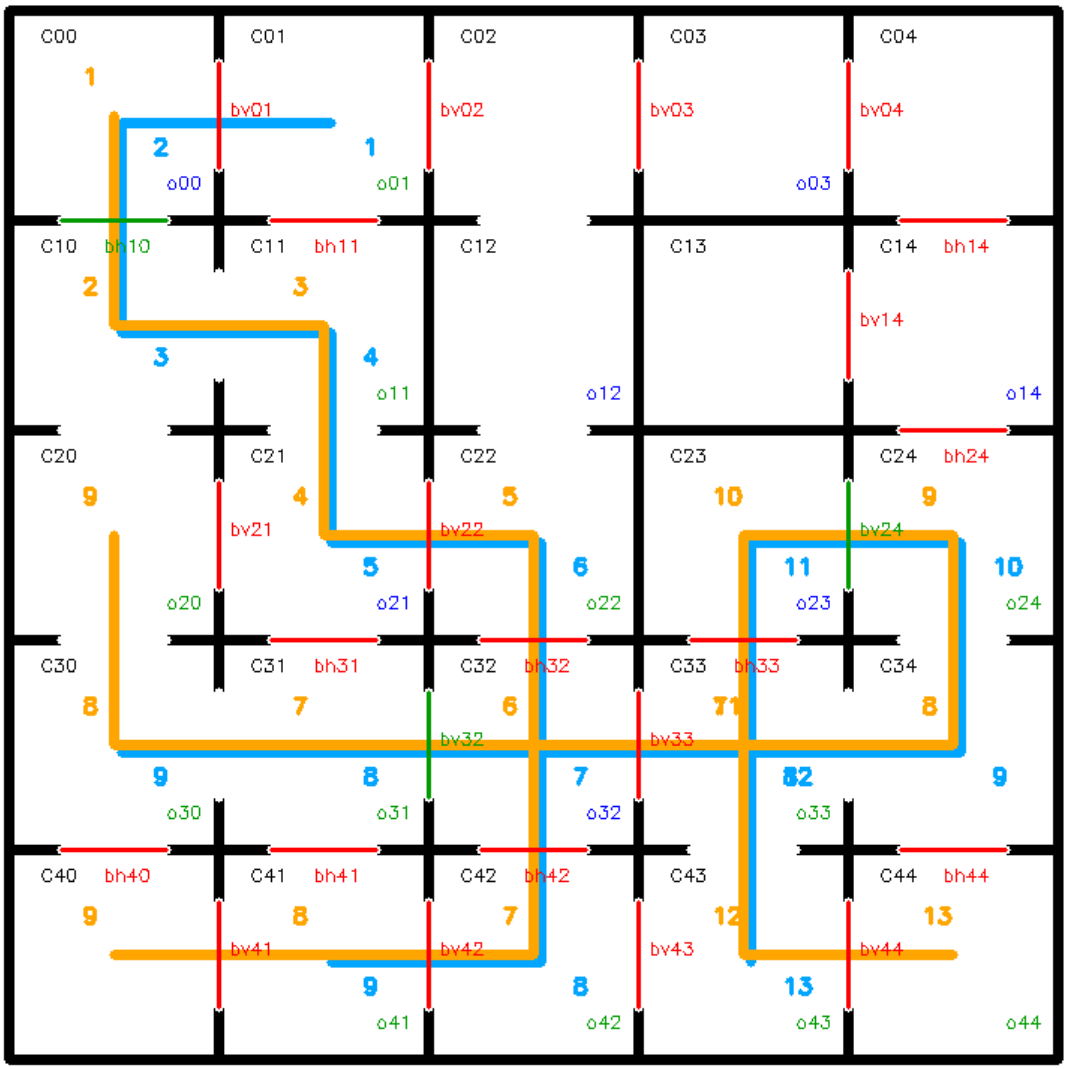
3.2.2.2 Joint Itineraries

Another element where the modification requires some care is in creating the joint itinerary. The joint itinerary represents the valid combined movement of all the agents. They can be formed in two ways: (i) furnished directly in joint-form as a problem parameter for MSSVI, or (ii) constructed via products of individual itineraries. The former case increases the modeling burden on the person providing the inputs, but is more expressive. It allows one to provide specifications where the behaviors of one of the agents is dependent on another, or to express some mutual interdependencies. One itinerary that exhibits such dependent behavior would be one where one agent must always be within one room of the other—a tracking or following problem. Figure 3.3 gives such an example along with its solution.

For case (ii), the time-stepping chosen (lockstep or independent) is an important variable as it determines the manner in which the individual itineraries are combined. For independent joint itineraries, unlike lockstep ones, self-loop transitions must be added into the individual itineraries before combining to give a joint itineraries. These self-loops encode that loitering by an individual agent is allowed. (As pointed out at the end of Section 3.1, quite apart from the selection of sensors, processing at runtime can catch the case where they all loiter indefinitely even if this was not part of the original specification, as this could be a mode (IV).)



((a)) ((b)) ((c)) ((d))



((e))

Figure 3.3: Example of a tracking problem where a leader (orange) starts in C00 and has a forked itinerary that can split into 3 different paths from C32, and a follower (blue) that must stay within one room of the leader. Inset (a)–(d) give this in detail: the leader's forked path is specified as (a), and depending on how the leader moves, the follower moves as (b), (c), or (d). The solution to this MSSVI appears in (e): green coloring of occupancy and beam sensors indicates they are in the minimal certifying sensor selection; the red/blue sensors are not needed.

3.3 SAT-Formulation of the Sensor Placement Problem

We re-formulate the sensor selection problem as a SAT problem by first converting all eight base constraints used for the ILP treatment of [7] into Conjunctive Normal Form (CNF) clauses. The combination of these translated clauses forms a SAT formula that can be directly used to determine whether the sensors available include a certifying selection, but may not give the optimal one. We then change the decision problem to that of finding a certifying sensor selection using at most k sensors via additional constraints. By iteratively decreasing k until no certifying selection can be found, we identify the minimum sensor count and associated placement.

For boolean variables, we will use a concise notation suggestive of an array: a letter followed by indices in brackets represents a single SAT variable.

$\mathbf{a}[\mathbf{q}|\mathbf{p}]$ is true if and only if state q in the product automaton can be reached from the initial state along a path that emits the same on-sensor events as the path from the initial state to the p state. Events for sensors that are turned off do not matter. This variable is a linchpin in the ensuring a certifying sensor selection, as to be certifying the sensors must be able to differentiate valid and non-valid paths, which they cannot do if the paths emit the same sensor events.

$\mathbf{b}[\mathbf{e}|\mathbf{y}]$ is true if and only if edge $e \in E$ triggers the sensor event $y \in Y$ and the sensor that emits $y \in Y$ is turned on.

The constraints are parameterized by elements in a product automaton with states $Q_{\mathcal{P}}$, constructed from the world graph and itinerary; each $q \in Q_{\mathcal{P}}$ is a pair of a world graph vertex and an itinerary state. Its final states are $F_{\mathcal{P}}$, and the transition function $\delta_{\mathcal{P}}(q, e)$ gives the state in $Q_{\mathcal{P}}$ reached when edge e is taken in the world graph. For more detail, the reader should see [7, Definition 4]; we have preserved their notation to help ease the task of cross referencing.

Using these, we can express the constraints.

Eight constraints follow, beginning naturally with rule for the (sole) initial state:

▷

$$\mathbf{a}[\mathbf{q_0^{\mathcal{P}}}|\mathbf{q_0^{\mathcal{P}}}] \qquad \text{(Eq. 3.1)}$$

This constraint establishes that the initial state of the product automaton is reachable from itself using two paths with the same sensor observations (none, for zero-length paths).

▷ For each $q \in F_{\mathcal{P}}, p \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}$,

$$\neg \mathbf{a}[\mathbf{q}|\mathbf{p}] \tag{Eq. 3.2}$$

Indicates that a final state in the product automaton cannot be reached from the initial state using a path with the same sensor observations as a path to a non-final state.

▷ For each $q \in Q_{\mathcal{P}} \setminus F_{\mathcal{P}}, p \in F_{\mathcal{P}}$,

$$\neg \mathbf{a}[\mathbf{q}|\mathbf{p}] \tag{Eq. 3.3}$$

This constraint is merely the reciprocal of (2).

▷ For each $e \in E$ and $y \in Y$ such that e does not trigger y,

$$\neg \mathbf{b}[\mathbf{e}|\mathbf{y}] \tag{Eq. 3.4}$$

Sets b variables to false if the edge e does not trigger the sensor event y. The b variables essentially encode the relation between edges and sensors.

▷ For each $e \in E$ and $y \in Y$ such that e triggers y,

$$(\mathbf{b}[\mathbf{e}|\mathbf{y}] \vee \neg \mathbf{u}[\eta(\mathbf{y})]) \wedge (\neg \mathbf{b}[\mathbf{e}|\mathbf{y}] \vee \mathbf{u}[\eta(\mathbf{y})]) \tag{Eq. 3.5}$$

For b variables where the edge e does trigger the sensor event y, they are true if the sensor that emits y is turned on and false otherwise.

Derivation note: the intermediate translation is $\mathbf{b}[\mathbf{e}|\mathbf{y}] = \mathbf{u}[\eta(\mathbf{y})]$, but $A = B$ can be expressed in CNF as $(A \vee \neg B) \wedge (\neg A \vee B)$.

▷ For each $q, p \in Q_{\mathcal{P}}$ and $e, d \in E$ such that $\delta_{\mathcal{P}}(q, e)$ and $\delta_{\mathcal{P}}(p, d)$ are both defined,

$$\neg\mathbf{a}[\mathbf{q}|\mathbf{p}] \vee \neg\mathbf{i}[\mathbf{e}|\mathbf{d}] \vee \mathbf{a}[\delta_{\mathcal{P}}(\mathbf{q}, \mathbf{e})|\delta_{\mathcal{P}}(\mathbf{p}, \mathbf{e})] \tag{Eq. 3.6}$$

If a state q is reachable with the same on-sensor observations as a state p (given by $\mathbf{a}[\mathbf{q}|\mathbf{p}]$ being true), and two edges e and d both emit the same on-sensor observations, then the states $\delta_{\mathcal{P}}(q, e)$ and $\delta_{\mathcal{P}}(p, d)$ can be reached with the same on-sensor observations. Along with (7) and (8), this constraint is responsible for a recursive definition of a values going back to the initial state value defined in (1).

Derivation note: we introduced a helper variable $\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}]$ to encode an equality, where for each $e, d \in E$ and $y \in Y$: $\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}] = (b_{e,y} \iff b_{d,y})$ which in CNF is:

$$\begin{aligned}\big(\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}] \vee \mathbf{b}[\mathbf{e}|\mathbf{y}] \vee \mathbf{b}[\mathbf{d}|\mathbf{y}]\big) \wedge \big(\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}] \vee \neg\mathbf{b}[\mathbf{e}|\mathbf{y}] \vee \neg\mathbf{b}[\mathbf{d}|\mathbf{y}]\big) \wedge \\ \big(\neg\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}] \vee \mathbf{b}[\mathbf{e}|\mathbf{y}] \vee \neg\mathbf{b}[\mathbf{d}|\mathbf{y}]\big) \wedge \big(\neg\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}] \vee \neg\mathbf{b}[\mathbf{e}|\mathbf{y}] \vee \mathbf{b}[\mathbf{d}|\mathbf{y}]\big)\end{aligned} \tag{Eq. 3.7}$$

To collapse a for-all clause we introduced another helper variable:

$$\mathbf{i}[\mathbf{e}|\mathbf{d}] = (\forall y \in Y, b_{e,y} = b_{d,y}) = \bigwedge_{y \in Y} \mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}]$$

which in CNF is

$$(\mathbf{i}[\mathbf{e}|\mathbf{d}] \vee \bigvee_{y \in Y} \neg\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}]) \wedge \bigwedge_{y \in Y} (\neg\mathbf{i}[\mathbf{e}|\mathbf{d}] \vee \mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}]) \tag{Eq. 3.8}$$

The intermediate translation is thus $a[q|p] \wedge i[e|d] \implies a[\delta_{\mathcal{P}}(q, e)|\delta_{\mathcal{P}}(p, e)]$. We then used

$A \wedge B \implies C$ being equivalent to $\neg A \vee \neg B \vee C$ to arrive at the final translated clause.

The combination of clause (6) along with the definition clauses above for $\mathbf{i}[\mathbf{e}|\mathbf{d}|\mathbf{y}]$ and $\mathbf{i}[\mathbf{e}|\mathbf{d}]$ forms the full SAT representation for constraint 6.

▷ For each $q, p \in Q_{\mathcal{P}}$ and $e \in E$ with $\delta_{\mathcal{P}}(q, e)$ defined,

$$\neg \mathbf{a}[\mathbf{q}|\mathbf{p}] \vee \neg \mathbf{j}[\mathbf{e}] \vee \mathbf{a}[\delta_{\mathcal{P}}(\mathbf{q}, \mathbf{e})|\mathbf{p}] \tag{Eq. 3.9}$$

If a state q is reachable with the same on-sensor observations as a state p (given by $\mathbf{a}[\mathbf{q}|\mathbf{p}]$ being true), and an edge e does not emit any on-sensor observations, then the states $\delta_{\mathcal{P}}(q, e)$ and p are reachable with the same on-sensor observations, since the observations for q and $\delta_{\mathcal{P}}(q, e)$ are therefore the same. Along with (6) and (8), this constraint is responsible for a recursive definition of a values going back to the initial state value defined in (1).

Derivation note: similar to in (6), we introduced a helper variable to collapse the for-all clause:

$$\mathbf{j}[\mathbf{e}] = (\forall y \in Y, b_{e,y} = 0) = \bigwedge_{y \in Y} \neg \mathbf{b}[\mathbf{e}|\mathbf{y}]$$

which in CNF is

$$(\mathbf{j}[\mathbf{e}] \vee \bigvee_{y \in Y} \mathbf{b}[\mathbf{e}|\mathbf{y}]) \wedge \bigwedge_{y \in Y} (\neg \mathbf{j}[\mathbf{e}] \vee \neg \mathbf{b}[\mathbf{e}|\mathbf{y}]) \tag{Eq. 3.10}$$

The intermediate translation is thus $\mathbf{a}[\mathbf{q}|\mathbf{p}] \wedge \mathbf{j}[\mathbf{e}] \implies \mathbf{a}[\delta_{\mathcal{P}}(\mathbf{q}, \mathbf{e})|\mathbf{p}]$. We then used $A \wedge B \implies C$ being equivalent to $\neg A \vee \neg B \vee C$ to arrive at the final clause.

The combination of clause (7) along with the definition clause above for $\mathbf{j}[\mathbf{e}]$ forms the full SAT representation for constraint 7.

▷ For each $q, p \in Q_{\mathcal{P}}$ and $e \in E$ with $\delta_{\mathcal{P}}(p, e)$ defined,

$$\neg \mathbf{a}[\mathbf{q}|\mathbf{p}] \vee \neg \mathbf{j}[\mathbf{e}] \vee \mathbf{a}[\mathbf{q}|\delta_{\mathcal{P}}(\mathbf{p}, \mathbf{e})] \qquad \text{(Eq. 3.11)}$$

This constraint is the reciprocal to (7), with the edge applied to p instead of q.

Derivation note: since the constraint is identical to (7) except for the right side of the implication, we can re-use the $\mathbf{j}[\mathbf{e}]$ variable we made earlier for (7).

So far, we have enough clauses to determine if there exists a certifying sensor selection (whether the SAT problem is satisfiable or not), and based on the outputted values for the $\mathbf{u}[\cdot]$ series variables, we get one of the certifying sensor selections, but it's not guaranteed to be the minimal selection.

In order to force the minimization and optimization, we add clauses to restrict the number of sensors that can be switched on, thereby turning it from a decision problem for whether there exists any certifying sensor selection, to a decision problem for whether there exists a certifying sensor selection $M \in S$ such that $|M| \leq k$.

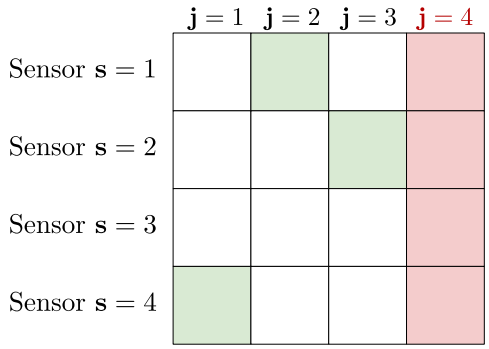


Figure 3.4: A visual representation of the additional SAT constraints using a grid scheme to restrict valid sensor activation configurations.

We introduce new variables $\mathbf{e}[\mathbf{i}|\mathbf{j}]$ for all $i \in \{1 \dots |S|\}$ and all $j \in \{1 \dots |S|\}$, forming a square grid of slots. Each row s corresponds to a sensor variable $\mathbf{u}[\mathbf{s}]$, such that the sensor is turned on only if one of the $\mathbf{e}[\mathbf{s}|\mathbf{j}]$ variables in the row is on. The formulation for this clause is

$$\mathbf{u}[\mathbf{s}] = \bigvee_{i \in \{1 \dots |S|\}} \mathbf{e}[\mathbf{s}|\mathbf{i}] \tag{Eq. 3.12}$$

By having a grid of slots, with each row corresponding to one sensor, and constraints such that only one slot can be turned on in each row and column, we can lock down columns of slots to force an arbitrary maximum number of sensors that can be turned on. This idea, including the concept of a "grid" can be seen in Figure 3.4. In this example the k upper bound is 3 sensors, locking down the 4^{th} column. Since only one cell per row and one cell per column is allowed to be on (green), that forces at most 3 sensors to be on but allows any combination of them. Here, sensors 1, 2, and 4 are online as an example placement.

To enforce only one slot on per row (each sensor can only be in one slot) we have for each $i \in \{1 \dots |S|\}$:

$$\sum_{j=1}^{|S|} \mathbf{e}[\mathbf{i}|\mathbf{j}] \leq 1 \tag{Eq. 3.13}$$

As this means no two $\mathbf{e}[\mathbf{i}|\mathbf{j}]$ can both be true, we can express this sum constraint in CNF as a conjunction of negated pairs (back with Boolean values):

$$\bigwedge_{a \in \{1 \dots |S|\}} \left(\bigwedge_{b \in \{a+1 \dots |S|\}} (\neg \mathbf{e}[\mathbf{i}|\mathbf{a}] \vee \neg \mathbf{e}[\mathbf{i}|\mathbf{b}]) \right) \tag{Eq. 3.14}$$

And, similarly, to enforce only one slot per column we have for each $j \in \{1 \dots |S|\}$: $\sum_{i=1}^{|S|} \mathbf{e}[\mathbf{i}|\mathbf{j}] \leq 1$, in CNF:

$$\bigwedge_{a \in \{1 \dots |S|\}} \left(\bigwedge_{b \in \{a+1 \dots |S|\}} (\neg \mathbf{e}[\mathbf{a}|\mathbf{j}] \vee \neg \mathbf{e}[\mathbf{b}|\mathbf{j}]) \right) \tag{Eq. 3.15}$$

To lock down the columns, we just force the variables for the slots in each of those columns

to false. If we want k maximum sensors on, we lock down each column $j \in \{k+1 \ldots |S|\}$:

$$\bigwedge_{i \in \{1 \ldots |S|\}} \neg \mathbf{e}[\mathbf{i}|\mathbf{j}] \qquad \text{(Eq. 3.16)}$$

When running the SAT instances incrementally, we simply add the constraint above for the new k value at each step.

We can run a series of iterations of this SAT problem, with decreasing k values starting from $|S|$ (the total number of sensors), until we reach an unsatisfiable instance. The k value and sensor selection set outputted from the prior SAT instance will then form the optimal solution to the MSSVI.

3.4 Graph Pre-Processing: Trimming

One issue with combining a series of world graphs into a single joint world graph is that the joint graph quickly explodes in size, growing exponentially as more agents are introduced. Having just two agents already squares the number of vertices and edges. In a 5×5 grid world example, the world graph has 70 edges, but with the addition of a second agent, the joint world graph has a whopping 8400 edges. To keep the runtime reasonable, we do some additional pre-processing on the graph. Depending on the itineraries, there may be sections of the graph that the agents can never legally visit. A drastic example is if a 5×5 grid environment is expanded into a 100×100 environment without altering the itinerary still mentions wandering about in the original 5×5 cells. The minimum sensor placement has not changed, so the extra rooms are irrelevant. Thus, we can safely prune any section of the graph that, once the agent reaches, must have already caused a violation in the itinerary. A caveat is that the efficacy of the trimming and the performance benefit is tied to the itinerary and can vary widely. An itinerary that describes visits of every single room in the environment, for example, would not yield any benefit from world graph trimming.

3.5 Attempt at Treatment of Multiple Itineraries

In certain cases we may wish to have multiple possible paths that a robot may follow, and to know which path it ended up taking. This scenario is multi-itinerary validation. Given two or more

itineraries by a single agent, we attempted to determine the minimum sensor selection required to not only validate that the agent has adhered to one of the itineraries, but also to identify which itinerary was adhered to. We identified two possible approaches to solving the multi-itinerary problem: the monolithic approach, where the itineraries are combined into a single problem, and the sequential approach, where each itinerary problem instance is solved by itself and the results are used to try and speed up the next problem instance. Our hope was that the sequential approach could produce solutions faster, for the trade-off of lower solution quality.

3.5.1 Monolithic Multi-Itinerary Validation

The idea behind the monolithic approach is akin to solving each itinerary validation problem independently, but combined in a single run. Since we are solving itinerary validation using SAT constraints, we can simply combine the constraints of each individual problem into a larger constraint set. The prior paper [7] proves the correctness of the solution outputted using the constraints for each single itinerary. Should we then combine constraints C_A and C_B for itineraries I_A and I_B to form C_{AB}, what the solver produces thus must satisfy the constraints in C_{AB}. As C_{AB} is a combination of C_A and C_B, it follows that said solution must satisfy the constraints C_A, so the sensor selection can distinguish I_A from $\neg I_A$. In other words, it can detect whether itinerary I_A was adhered to or not. The same argument indicates the same for I_B. Hence, the solution can distinguish any combinations of the itineraries. The constraints for each single-itinerary problem can be broken down into the ones relating to the product automaton and thus are specific to the itinerary, and the rest, which can be shared. For example, sensor definition constraints are not itinerary-specific and can be defined once, instead of defining a set for each itinerary. Figure 3.5 shows the solutions for monolithic and the individual itineraries improving as runtime increases. The monolithic method works well, but we were more interested in the possible benefits from the sequential approach.

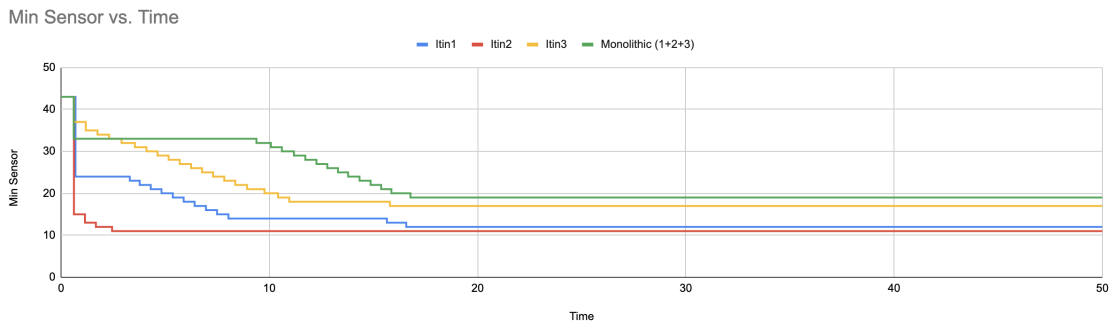


Figure 3.5: Stepping chart comparing min-sensor solutions for each individual itinerary to the monolithic multi-itinerary solution as runtime increases. Large drops are due to the SAT solver finding a better solution than the k-value it was asked for. See Section 3.3 for details on stepping.

3.5.2 Sequential Multi-Itinerary Validation

The sequential approach involves solving each itinerary problem instance and then using the results to try and speed up the next itinerary instance. We do this by "locking in" the sensors required for the previous itineraries, forcing them to be on. By locking in the sensors, we reduce the choices available and therefore allow a speedup for solving subsequent itineraries, but at the expense of solution quality since the best overall solution may not have those sensors turned on. When attempting the sequential approach, we were unable to achieve solution times better than the monolithic approach, mostly due to a key issue with the SAT solver. The solver, in some cases, tends to get stuck at the very last SAT step where the constraints are supposed to be unsatisfiable. Instead of returning SAT/UNSAT, it will run indefinitely. We were unable to find a proper way around this issue, despite trying different solvers, other than imposing a propagation limit as a timeout. This limit, however, can cause early exits (leading to higher sensor counts reported as the minimum) and disproportionately affects the sequential approach. Since the sequential method runs each itinerary instance in sequence, it has to iterate through the stepping-down algorithm several times, potentially getting stuck on each itinerary until hitting the timeout. The monolithic method only runs one large problem instance, and as such has fewer timeouts than the sequential. Since we were unable to improve the sequential approach due to this solver quirk, we instead focused our efforts on multi-agent itinerary validation.

4. EXPERIMENTAL RESULTS

4.1 Experiments

To examine how the SAT-based implementation fares in comparison to the original ILP approach, we ran two experiments. The first compared SAT and ILP for a single agent in a corridor case study and the second compared SAT and ILP for multiple agents in a grid world. Both of these types of worlds are parametrizable so that we can vary their size to examine scaling properties with ease. The goal was to illustrate any performance benefits that the SAT approach brings, and whether it is more applicable than ILP to problems that rapidly scale in size.

4.2 Single-agent Comparison

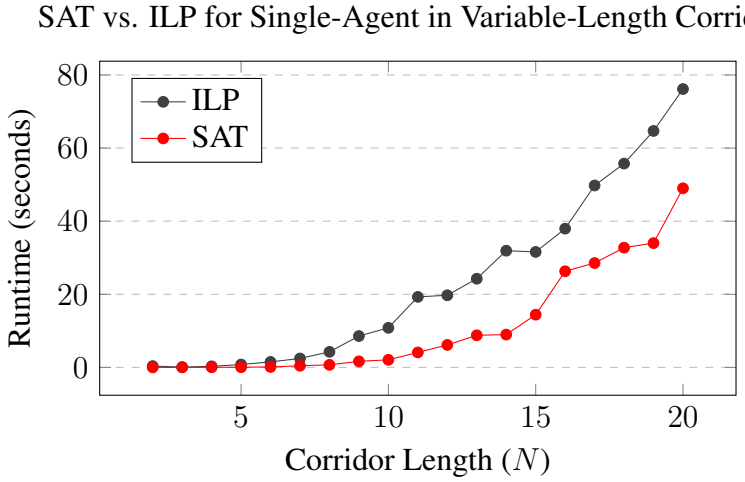


Figure 4.1: Comparison of ILP versus SAT performance for a **single agent** in N-length corridors, each an itinerary traversing the entirety of the corridor. Each data point represents the average of three runs.

The corridor case study involves a variable length corridor of rooms. Each room has a single room attached to it perpendicular to the corridor, similar to a closet. Occupancy and beam sensors are distributed throughout the rooms. The single agent is instructed to travel down the entire corridor, traversing all the rooms. We varied the corridor length from 2 to 20 rooms long,

and for each corridor length we ran three runs each for SAT and ILP and took the average runtime as the final result for the data point.

Both plots in Figure 4.1 exhibit rapid growth due to how the base constraints require looping over all pair combinations of edges, but the SAT approach performed better than ILP, with generally about half the runtime. For the lengths shown here, SAT could solve a problem size about 3–5 rooms longer for the same runtime as ILP.

4.3 Multi-agent Comparison

The second experiment used the grid world case study, where the grid has 3 columns and a variable number of rows. The important difference is that there are now two agents moving through the world, each with their own itinerary. With two agents, the joint world graph grows quadratically with more rows in the grid. The agent itineraries are pre-generated zig-zag paths, and are always the same for each run on the same grid size. The data points in Figure 4.2 are each an average over five runs. The plot also compares the lockstep and independent time cases, showing how the extra degrees of freedom present in independent itineraries result in longer runtimes.

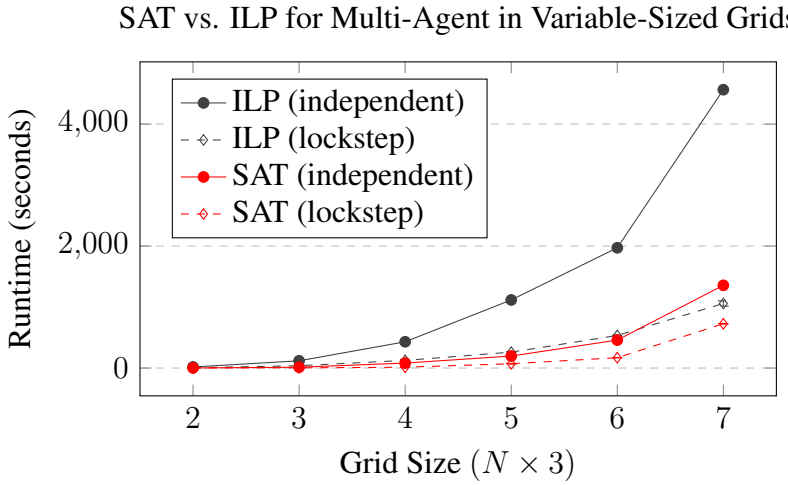


Figure 4.2: Comparison of ILP versus SAT performance for **two agents** in $N \times 3$ grids of varying row count, each with zig-zag agent itineraries. Itineraries are always identical between SAT and ILP runs of the same N. Each data point represents the average of five runs. Error bars are drawn for the lockstep cases showing ± 1 standard deviation, though barely visible.

The performance difference is much more evident for the multi-agent experiment. The ILP runtimes grow rapidly and quickly reach the point where it takes an unworkable amount of time to run, requiring over an hour for the 7×3 grid compared to 20 minutes for SAT. The difference shown in the single-agent experiment is greatly exacerbated here due to the multi-agent world graph growing quadratically in size, compounded by the ILP and SAT constraints growing quadratically relative to the world graph. It is clear that for problems where the complexity escalates, the SAT-based approach is superior.

4.4 Graph Trimming

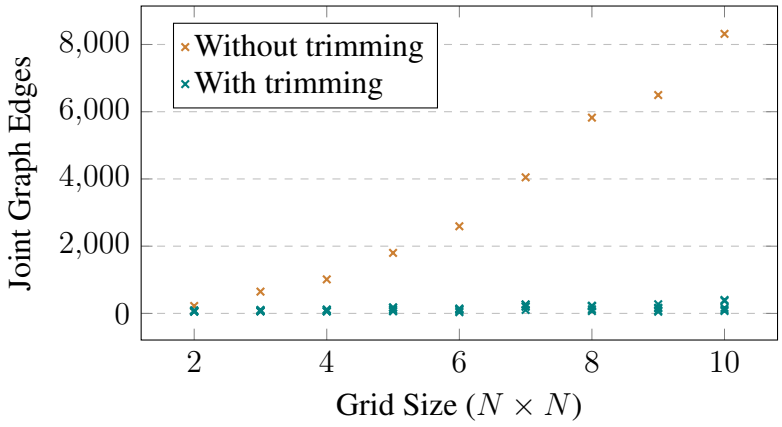


Figure 4.3: Demonstration of trimming effectiveness for varying grid sizes N for $N \times N$ grids, each with 5 randomly-generated agent itineraries of varying lengths.

Consider the extent to which trimming the world graph can affect runtime. Figure 4.3 shows a scatterplot of joint graph edge counts before and after trimming, with multiple random itineraries as trimming effectiveness depends on the itinerary in question. We see that graph sizes are drastically different. The data in Figure 4.4 table show its effect on computation time: there is an N^2 relation to the world graph edges. The seemingly small difference in Figure 4.3 for the 3×3 grid is a difference in runtime by a factor of over 200. Evidently, solution times for multi-agent scenarios would become infeasible without graph trimming.

Grid Size	Trimmed	Not trimmed
3×3	$2.462\,\mathrm{s}$	$536.551\,\mathrm{s}$
4×4	$3.251\,\mathrm{s}$	—
5×5	$7.604\,\mathrm{s}$	—
6×6	$11.574\,\mathrm{s}$	—
7×7	$15.163\,\mathrm{s}$	—

Figure 4.4: Runtimes before and after trimming for multiple grid sizes.

5. CONCLUSION AND FUTURE WORK

We have explored a variation of the sensor selection problem with multiple agents in the same environment, and in the process examined a more efficient SAT-based reformulation of the problem, treatments of time and interleaving in itineraries, and a graph trimming optimization. These new formulations provide the tools to handle more complex environments and interactions, as well as the performance speed-ups required to handle the larger problem instances.

In our experiments we treated scenarios of two agents. Although the formulation holds for any number of agents, a trial of three lockstepped agents on a 4×3 grid took 13 minutes compared to the 13 seconds required for two agents, with the same setup as in Figure 4.2. Some factor in-crease in runtime is unavoidable, but we believe opportunities remain for further optimizations, including more aggressive trimming and approximation algorithms. Considering the marked im-pact that graph trimming has had, more examination of it is warranted. As is the nature of NP-hard problems, there likely remains more to be done.

REFERENCES

[1] P. Zhong and D. A. Shell, "Sensor Selection for Behavior Validation of Multiple Agents." under submission to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022.

[2] I. Bukhori, Z. Ismail, and T. Namerikawa, "Detection strategy for kidnapped robot problem in landmark-based map Monte Carlo Localization," in *International Symposium on Robotics and Intelligent Sensors (IRIS)*, pp. 75–80, 2015.

[3] J. O'Rourke, *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[4] L. Chen and C. D. Nugent, *Human activity recognition and behaviour analysis*. Springer, 2019.

[5] S. Balandin and H. Waris, "Key properties in the development of smart spaces," in *International Conference on Universal Access in Human-Computer Interaction (UAHCI)*, pp. 3–12, 2009.

[6] D. Uchechukwu, A. Siddique, A. Maksatbek, and I. Afanasyev, "ROS-based integration of smart space and a mobile robot as the internet of robotic things," in *Conference of Open Innovations Association (FRUCT)*, pp. 339–345, 2019.

[7] H. Rahmani, D. A. Shell, and J. M. O'Kane, "Sensor selection for detecting deviations from a planned itinerary," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6511–6518, 2021.

[8] X. Yin and S. Lafortune, "A general approach for optimizing dynamic sensor activation for discrete event systems," *Automatica*, vol. 105, pp. 376–383, 2019.

[9] J. Yu and S. M. LaValle, "Story validation and approximate path inference with a sparse network of heterogeneous sensors," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4980–4985, 2011.

[10] B. Tovar, F. Cohen, L. Bobadilla, J. Czarnowski, and S. M. Lavalle, "Combinatorial filters: Sensor beams, obstacles, and possible paths," *ACM Transactions on Sensor Networks*, vol. 10, no. 3, pp. 1–32, 2014.

[11] A. Censi, "A Class of Co-Design Problems With Cyclic Constraints and Their Solution," *IEEE Robotics and Automation Letters*, vol. 2, pp. 96–103, Jan. 2017.

[12] D. A. Shell, J. M. O'Kane, and F. Z. Saberifar, "On the design of minimal robots that can solve planning problems," *IEEE Trans. on Automation Sci. and Engineering*, vol. 18, no. 3, pp. 876–887, 2021.

[13] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3 ed., 2006.