PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

PLANNING AND GOAL RECOGNITION WITH SAT-BASED APPROACHES: A SURVEY

KIN MAX PIAMOLINI GUSMÃO

Monograph submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Felipe Meneguzzi

Porto Alegre
2023

ACKNOWLEDGMENTS

I want to thank my advisor, professor Felipe Meneguzzi for all the support during the development of this survey, and for allowing me to use figures and practical examples from his lecture slides in this survey.

PLANEJAMENTO E RECONHECIMENTO DE OBJETIVOS COM ABORDAGENS BASEADAS EM SAT: UMA PESQUISA

RESUMO

O reconhecimento de objetivos e planos tem sido o foco de muitas pesquisas na área de inteligência artificial e planejamento automatizado, uma vez que as aplicações são diversas, como cuidado a idosos, detecção de intrusão, entre outras. Complementarmente, o campo de reconhecimento de objetivos em ambientes multiagente tem crescido muito nos últimos anos, pois em muitos casos não há um único agente atuando, mas um grupo de agentes no mesmo ambiente que podem ou não estar cooperando em times em direção a um objetivo comum ou a múltiplos objetivos. Existem muitas soluções para problemas de reconhecimento de objetivos, em cenários de agente único e multiagente, que se baseiam em técnicas de planejamento e na teoria do domínio de planejamento. No entanto, há pouco trabalho no campo para reduzir um problema de reconhecimento de objetivos a um problema SAT e usar um resolvedor de problemas SAT padrão para resolvê-lo. Enquanto isso, esse tipo de solução é comum no campo de planejamento automatizado. Nesta pesquisa, analisamos o embasamento teórico e as soluções relevantes nas áreas de planejamento e reconhecimento de objetivos de agente único e multiagente, e o uso de soluções baseadas em SAT nessas áreas de pesquisa.

Palavras-Chave: inteligência artificial, planejamento, planejamento multiagente, planejamento automatizado, reconhecimento de objetivos, reconhecimento de objetivos multiagente, satisfatibilidade booleana.

PLANNING AND GOAL RECOGNITION WITH SAT-BASED APPROACHES: A SURVEY

ABSTRACT

Goal and plan recognition have been the focus of much research within the field of artificial intelligence and automated planning, since the applications are manyfold, such as elder care, intrusion detection, among others. Complementary, the field of multiagent goal recognition has grown a lot in the past years, as in many cases there is not a single agent acting, but a group of agents on the same environment that may or may not be cooperating as a team towards a common goal or towards multiple different goals. There are many solutions for goal recognition problems, in both single and multiagent scenarios, that leverage on planning techniques and planning domain theory. However, there is little work on the field on reducing a goal recognition problem to a SAT problem and using a standard SAT solver to solve it. Meanwhile, this kind of solution is common in the automated planning field. In this survey, we go over the theoretical background and relevant solutions on both single-agent and multiagent planning and goal recognition areas, and the usage of SAT-based solutions in those research areas.

Keywords: artificial intelligence, planning, automated planning, multiagent planning, goal recognition, multiagent goal recognition, boolean satisfiability.

LIST OF FIGURES

LIST OF ACRONYMS

CDCL – Conflict-Driven Clause Learning

CMAP – Centralized Multi-Agent Planning

FD – Fast-Downward Planning System

FF – Fast-Forward Planning System

HSP – Heuristic Search Planner

MAGR – Multiagent Goal Recognition

MAP – Multiagent Planning

MAPR – Multi-Agent Planning by Plan Reuse

MAPRAP – Multiagent Plan Recognition as Planning

MA-PDDL – Multiagent PDDL

MA-STRIPS – Multiagent STRIPS

OMT – Optimization Modulo Theory

PDDL – Planning Domain Definition Language

P-MAPRAP – Probabilistic Multiagent Plan Recognition as Planning

PMR – Plan Merge by Reuse

RPG – Relaxed Planning Graph

SAT – Boolean Satisfiability Problem

SMT – Satisfiability Modulo Theory

STRIPS – Stanford Research Institute Problem Solver

VSIDS – Variable State Independent Decaying Sum

CONTENTS

1. INTRODUCTION

Correctly inferring an agent's goal based on observations over their actions has multiple real-world applications [39]. A few examples are elder-care applications [15], intrusion detection systems [16], exploratory domain models [37, 40], among many others. Additionally, there are many cases where we might not want to identify the intent of a single agent, but of a group of agents who might or might not be cooperating towards a common goal.

Recent approaches to both single-agent and multiagent goal recognition have managed to obtain good accuracy in the recognition process using different approaches [44, 42, 1, 2, 56, 57]. The use of SAT solvers for goal recognition tasks has been, however, scarce over the years, with the most prominent example being the use of Weighted MAX-SAT by Zhuo et al. for multiagent scenarios [55, 56, 57]. This differs from automated planning, the overarching area, which extensively uses satisfiability approaches [23, 26, 12, 47, 11].

Given the performance of SAT-based approaches to solve planning problems that allow multiple parallel actions [47], we aim to further explore the usage of SAT-based solutions for planning and goal recognition applications in both single-agent and multiagent scenarios. Our goal with this survey is to revisit the theoretical background around both single-agent and multiagent planning and goal recognition and the usage of SAT-based solutions to solve problems in those fields.

We organize the remainder of this survey as follows. First, in Chapter 2, we revisit the formalism and notations, the problem description languages, and some frameworks and algorithms used in single-agent automated planning. In Chapter 3, we revisit the theoretical background around the Boolean Satisfiability Problem (SAT), and the way that SAT and planning fields merge, going over the different problem encodings, SAT-solving strategies used in planning applications, and some single-agent planners that rely on SAT-based approaches. In Chapter 4, we discuss multiagent planning, its formalism and notations, problem description language, and some frameworks and algorithms. In Chapter 5, we discuss single-agent goal recognition, its formalism and notations and some frameworks. In Chapter 6, we discuss multiagent goal recognition, its formalism and notations, and some frameworks and algorithms. Finally, in Chapter 7, we summarize our survey and discuss our final remarks.

2. AUTOMATED PLANNING

Automated planning or simply planning is the task of obtaining a sequence of actions (a plan) that can achieve a goal state starting from some initial state [17]. Figure 2.1 depicts an example of a planning environment with multiple possible goals, where we might want to derive a plan to direct our robot agent to one of the goals. Planning has multiple applications in many fields, such as video-games AI [27], autonomous systems [21], and many others. In this Chapter, we outline the automated planning field by going over the formalism and notation, in Section 2.1, the main languages for describing problems, in Section 2.2, and the main frameworks and algorithms, in Section 2.3.

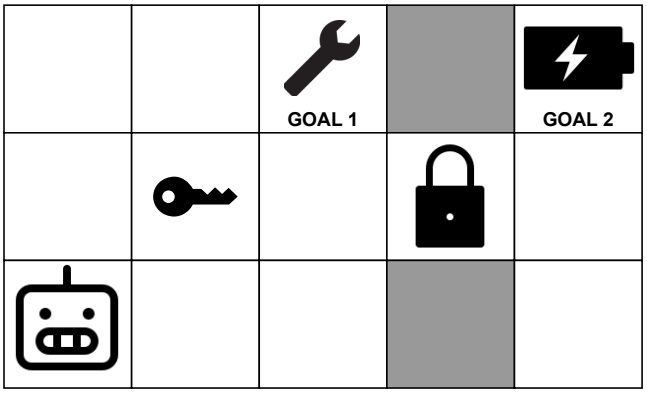

Figure 2.1 – Planning environment example

2.1 Formalism and Notation

There are two basic structures in planning: facts and actions. Facts are ground predicates associated to zero or more terms ($\tau_1, \tau_2, ..., \tau_n$). If we take the well-known planning domain *Blocks-World* as an example [32], we could have a fact `(on a b)`, where the predicate `on` is associated to the terms `a` and `b`, representing the fact that block `a` is on top of block `b`, or `not (on a b)`, representing the fact that that block `a` is not on top of block `b`. A *state* comprises a conjunction of facts and describes the environment state at a given time step.

Actions are operators instantiated to free variables, where each variable represents an object of the environment. An operator a is a tuple $a = \langle name(a), pre(a), eff(a) \rangle$, where $name(a)$ is the name or description of the operator, $pre(a)$ are the preconditions of the operator, *i.e.* facts that must be true before the operator is executed and $eff(a) = eff(a)^+ \cup eff(a)^-$ are the effects of the operator, where $eff(a)^+$ is an add-list of facts, *i.e.* facts that become true once the operator executes, and $eff(a)^-$ is a delete-list of facts, *i.e.* facts that become false once the operator executes.

In *Blocks-World*, we obtain the action `(unstack a b)` by instantiating the operator `unstack` to the variables `a` and `b`, representing the action to remove block a from the top of block b. We say that an action is applicable at a given state S if its preconditions are satisfied in that state, *i.e.* $\forall p \in pre(a), p \in S$. When an agent performs an action at a given state S, it generates a new state S', where $S' = S \backslash eff(a)^- \cup eff(a)^+$.

A classical planning domain definition is a tuple $\Xi = \langle \Sigma, \mathcal{A} \rangle$, where Σ is a finite set of ground facts and $\mathcal{A}$ is a finite set of ground actions, *i.e.* all facts and actions possible to occur in the planning domain. A planning instance is a triple $T_P = \langle \Xi, \mathcal{I}, G \rangle$, where Ξ is the planning domain definition, $\mathcal{I}$ is the initial state, and G is the goal state. Finally, a plan is a sequence of actions $\pi = \langle a_1, a_2, ..., a_n \rangle$ that modify the initial state I into the goal state G.

2.2 Problem Description Languages

The first standardized language for describing planning problems is STRIPS, named after that planner that first used it, the Stanford Research Institute Problem Solver [14]. Using STRIPS, one can define the initial and goal states and a set of operators. Each operator has a name, a set of preconditions, a set of effects and, optionally, a set of variable constraints.

For years, STRIPS was the standard language for defining planning problems, until the Planning Domain Definition Language (PDDL) was introduced in 1998 [36]. Currently, PDDL is the standard language for defining planning domains and problems. It is mostly derived from STRIPS, maintaining all of its features, while having greater description capabilities that allows one to define such things as object types, negated preconditions, conditional effects, among other features, being considered an evolution to STRIPS.

The definition of a full planning instance in PDDL is comprised of two files: a domain file and a problem file. The domain file contains the information needed to define a planning domain: a list of fact predicates and a list of action operators. As previously established, the list of facts is a list of predicates related to zero or more terms. In contrast, the list of actions is a list of operators with defined preconditions and effects. Listing 2.1 depicts an example of a simple definition of the *Blocks World* planning domain.

```
1 (define (domain blocks)
2
3  (:requirements :strips)
4  (:constants-def table)
5
6  (:predicates (on ?a ?b)
7               (block ?b)
8               (clear ?b) )
9
```

```
10  (:action move
11   :parameters (?b ?x ?y)
12   :precondition (and (on ?b ?x)
13                      (clear ?y)
14                      (clear ?b)
15                      (block ?b)
16                      (block ?y)
17                 )
18
19         :effect (and (on ?b ?y)
20                      (clear ?x)
21                      (not (on ?b ?x))
22                      (not (clear ?y))
23                 )
24  )
25
26  (:action moveToTable
27   :parameters (?b ?x)
28   :precondition (and (on ?b ?x)
29                      (clear ?b)
30                      (block ?b)
31                      (clear ?x)
32                 )
33
34   :effect (and (on ?b table)
35                (clear ?x)
36                (not (on ?b ?x))
37           )
38  )
39 )
```

Listing 2.1 – Simple *Blocks World* domain definition

The file defines a list of predicates applied to variables, preceded by the "?" symbol, as the ground predicates, or facts, for the domain. We see predicates to verify that a given block is on top of another given block, to check whether a given object is a block and to check whether a block is clear, *i.e.* if there are no blocks on top of it.

The definition also contains a list of operators, each denoted by the reserved word "action". As in the formal definition, each action definition must include an action name, the list of variables that action takes, a list of preconditions regarding the variables, in the form of a conjunction of facts, and a list of effects, also as a conjunction of facts. The positive facts represent the add-list of effects, while the negated ones represent the delete-list.

Having the domain defined, we can then define the planning instance, or the problem. The problem must be defined in a separate problem file, containing information

such as the objects over which the facts and actions from the domain will be applied, the initial state and the goal state. Listing 2.2 depicts a problem definition for the previously defined domain.

```
1  (define (problem pb1)
2   (:domain blocks)
3   (:objects a b c table)
4   (:init (on a table)
5          (on b table)
6          (on c a)
7          (block a)
8          (block b)
9          (block c)
10         (clear b)
11         (clear c))
12  (:goal (and (on a b)
13              (on b c)
14         )
15  )
16 )
```

Listing 2.2 – *Blocks World* problem definition example

The first thing in the problem definition must be the problem name, defined in line 1. The next definition is the planning domain of the problem, as defined in line 2. It must also define the list of objects in the instance, as defined in line 3, the initial state, and the goal state. The initial state (line 4) is defined by a list of facts over the defined objects, while the goal state (line 12) is a conjunction of facts over the objects. Figure 2.2 illustrates the initial and goal states defined in Listing 2.2.

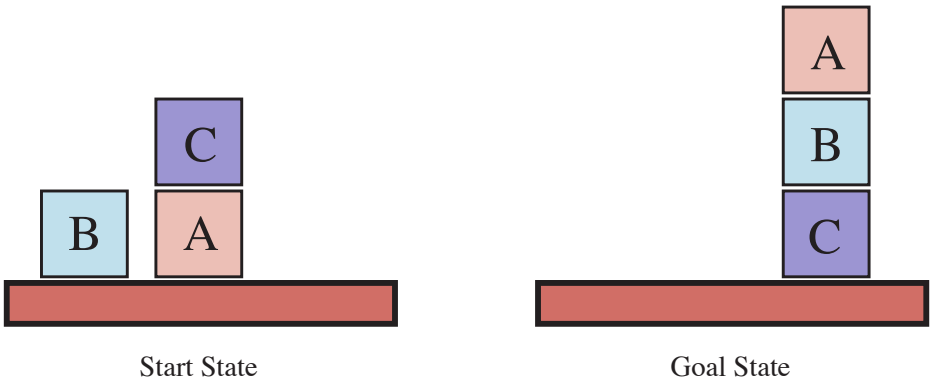

Figure 2.2 – *Blocks World* initial and goal state examples.

2.3 Frameworks and Algorithms

The first and simplest algorithms for solving planning problems use forward or backward search strategies. STRIPS is one of the first and better known algorithms at the beginning of planning research [14]. Like its predecessors, STRIPS uses backward search, but reduces the search space by applying some strategies. One of these strategies is only considering sub-goals that are preconditions to the last operator added to the plan and committing to execute a given operator, not backtracking over this commitment if the current state satisfies all the operator's preconditions. Apart from being one of the first algorithms to apply planning-specific strategies to planning algorithms, STRIPS also defined the first standard language for defining planning problems.

Years later, another planner would become a landmark in the automated planning research field. The Graphplan planner [4] uses a graph to annotate problem information, which helps prune sub-goals in the backward search done in the solution phase of the algorithm. In this graph, every even level is a fact level, where each node represents a ground fact, composing the state at that time step. Complementary, every odd level is an action level, where each node represents a ground action that is applicable at that time step, *i.e.* its preconditions are satisfied at that time step.

The first level contains every fact in the initial state, while the last level contains every fact in the goal state. Every fact node connects to the actions in the next level to which it is a precondition, while every action node connects to the facts in the next level that are an effect for that action. We call this structure a *planning graph*. Figure 2.3 illustrates the general structure of a planning graph.

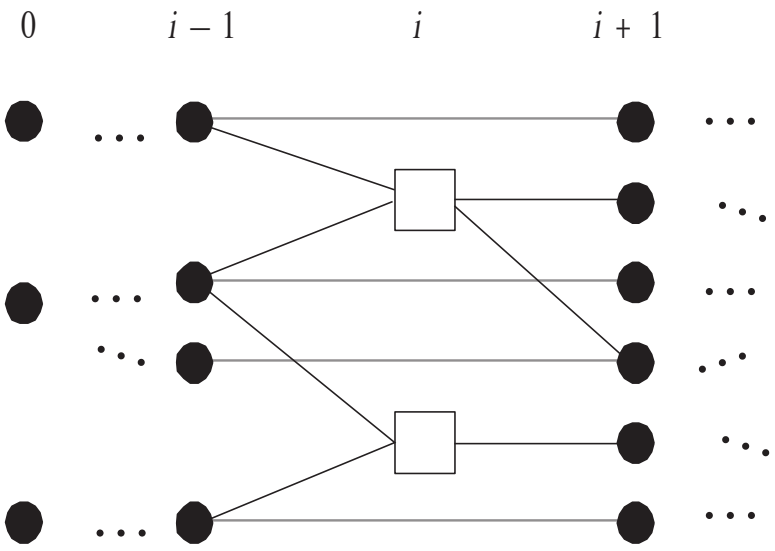

Figure 2.3 – Planning graph general structure

With the planning properly graph set up, Graphplan then starts the search process. The planner performs a backward-chaining level-by-level strategy. First, it starts by analyzing the last level of the graph (for convenience, we call the current level t), searching for the goal state predicates. It then goes to level $t-1$, searching for the actions that have those predicates as add-effects. Once it finds those actions, it takes the preconditions for these actions as sub-goals, and the recursive search process goes to the level prior to that, treating it as level t now. The planner repeats this process until it reaches the first graph level, then it derives a plan from the actions selected to fulfill the sub-goals at each recursion. If an action fails to fulfill a sub-goal, Graphplan will try to select another one until that sub-goal is fulfilled. If it comes to a point that some sub-goal from level t could not be fulfilled by any actions in level $t-1$, the planner returns a failure, as the goal is not achievable.

There is also a relaxation of the planning graph structure where the delete-list of effects is ignored in the relation between the facts and actions. We call this relaxed structured a *relaxed planning graph* (RPG). Graphplan brought major improvements to the planning process that are still referenced in recent works, such as the ability to represent mutual exclusion between actions, the possibility of having parallel actions when they are not mutually exclusive, memoization capabilities obtained by having the actions fixed at their respective time steps, and not reducing instantiations at search time, as the planner generates the entire graph before the search process begins.

Other automated planning approaches leverage on heuristic search, such as the Heuristic Search Planner (HSP) [5] and the Fast-Forward planning system (FF) [20]. Both use heuristic search based on a delete-free relaxation of planning problems, where one ignores the delete-list of effects.

HSP was one of the planners in the AIPS 98 planning competition [32], and it showed that heuristic-based planners can be competitive against the existing Graphplan [4] and SAT planners such as BLACKBOX [24]. The work that introduced HSP [5] presents three variations of HSP. The first one is a hill-climbing planner with additive heuristic that, even though showed to be competitive against the other planners in the competition, was not optimal nor complete. To solve the completeness issue, they presented a variation called HSP2, that uses best-first search instead of hill-climbing, and it not only showed to be better at solving the planning problems, but also obtained a better time performance than the hill-climbing version. Finally, they present a third version, called HSPr, which does a backward search instead of the forward one performed by the first two variants, performing a regression from the goal state. They discuss that this prevents the full heuristic recomputation at every single state, resulting in a much better time performance.

The influence of HSP in the field derived the Fast-Forward planning system (FF) [20]. FF is widely based on HSP, but has a major increase in performance, being the best

planner in the AIPS 2000 planning competition [3]. The main differences from HSP are the heuristic, that takes into account positive interactions between facts, using enforced hill-climbing instead of hill-climbing for a search method, and pruning action nodes based on identifying the ones that are more helpful on reaching the goal.

These planners kept deriving more high-achieving planners, such as the Fast-Downward planning system (FD) [19] and the LAMA planner [45]. FD is also a forward search heuristic planner, but it translates the planning task into what the call a multivalued planning task, and solve it by hierarchically decomposing the task to compute the heuristic function. The Fast-Downward planning system was the best performing planner in the classical track of the 4th International Planning Competition at ICAPS 2004. A few years later, the LAMA planner appeared, based on the FD planning system, but using the FF heuristic combined with a heuristic based on planning landmarks, *i.e.* necessary facts or actions that must be achieved or executed to achieve a given goal. LAMA was the best performing planner in IPC 2008's sequential satisfying track.

3. PLANNING AS SATISFIABILITY

One of the approaches to solving a planning problem is converting it into a SAT instance. For this, one must properly encode the planning problem into a SAT problem, which allows solving the planning problem with a regular SAT solver. In this Chapter, we go over the theoretical background involving the Boolean Satisfiability Problem (SAT) and some solving techniques, in Section 3.1, different encodings for planning as satisfiability, in Section 3.2, different approaches in SAT solving for planning, in Section 3.3, and some planners based on SAT, in Section 3.4.

3.1 Boolean Satisfiability Problem (SAT)

The Boolean Satisfiability Problem (SAT) is the problem of checking whether a given boolean formula is satisfiable or not. In other words, we need to check if there exists a model, *i.e.*a truth-value assignment for each variable in the formula, that makes that formula true. An example is the formula below:

$$(p \vee \bar{q} \vee r) \wedge (\bar{p} \vee \bar{q} \vee \bar{r}) \wedge (\bar{p} \vee q \vee \bar{r})$$

The formula above is satisfiable, and a valid model for it is $\{p = \mathit{False}, q = \mathit{False}, r = \mathit{True}\}$. By contrast, the formula below is an example of an unsatisfiable formula, as no model exists that makes the formula true:

$$(p \vee q) \wedge (p \vee \bar{q}) \wedge (\bar{p} \vee q) \wedge (\bar{p} \vee \bar{q})$$

This is probably the most widely studied problem in computational logic, as it was the first problem to be proven NP-complete, through the Cook-Levin theorem [8, 31]. This not only means that SAT is in NP time complexity class, *i.e.* it can be solved in polynomial time by a non-deterministic Turing Machine, but also means that any problem in NP can be reduced in polynomial time to a SAT problem. This is extremely useful, as we don't need to know how to solve every problem in NP. We can just convert it to a SAT problem and solve it with a standard SAT solver.

One of the first procedures to be used in SAT solving is the Davis-Putnam procedure [10, 9], or DPLL, named after the initials of its creators. It takes a boolean formula as input and outputs a boolean value that represents whether the formula is satisfiable or not, and the found model that satisfies the formula, in case it is satisfiable. Some implementations only return the boolean value, but we assume implementations that also return the model, as the algorithm is capable of doing so. The algorithm works as a re-

cursion over the formula. It starts by propagating all the unit clauses, *i.e.* it selects the variables that appear alone in clauses and assigns truth-values to them to satisfy those clauses. This is called the *unit propagation* step. It also removes from the formula all the clauses that were satisfied by this assignment. Finally, it selects a variable to assign a truth-value to, and applies that value to the formula. It recursively tries to assign the value “true”, and then “false”. The variable selection depends on the implementation, and can be as simple as a random selection, or have a more clever procedure such as selecting the variable with most occurrences among the clauses. On the next recursion, the unit propagation step will remove this newly added unit clause with the chosen variable, and every clause satisfied by it. If all variables in a clause are assigned, but the clause is not yet satisfied, the clause is called an *empty clause*, still remaining in the formula. Conversely, when a clause is satisfied, it is removed from the formula. Each recursion, the algorithm checks if the formula is empty, what makes it trivially satisfied, and also checks for the presence of any empty, *i.e.* unsatisfiable, clauses. If there is an empty clause, the formula is not satisfied with that model, and it backtracks to the last recursion to choose a different variable assignment.

Although the DPLL procedure is still widely used, the current state-of-the-art procedure for SAT solving is an algorithm commonly called Conflict-Driven Clause Learning (CDCL), originally developed in [34, 35], under the name GRASP. The CDCL algorithm uses DPLL as a base, but adds major improvements to it. For example, it adds clause conflict learning, as it can learn exactly which truth-value attribution caused the conflict that led to the formula being unsatisfiable with that model. Furthermore, it adds non-chronological backtracking. The DPLL procedure can only backtrack to the previous recursion level each time, which may lead to the algorithm still exploring paths that will lead to unsatisfiability. The clause learning performed by CDCL allows it to backtrack directly to the assignment that led to the conflict. Other improvements may vary depending on the implementation, such as analyzing isolated clauses separately, in parallel, and using heuristics to make a better choice on the variable to assign next.

As a heuristic example, we take the one used in Madagascar SAT planner [47] to improve SAT solving performance: the Variable State Independent Decaying Sum (VSIDS) heuristic, first implemented as a part of the Chaff SAT solver [38]. VSIDS is a heuristic to improve literal choice in the current recursion of the CDCL algorithm. It works by computing a score for each variable, then CDCL chooses the one with the highest heuristic score. The initial score is the number of literal occurrences of the variable. Then, for each new conflict clause found, the score is incremented for all the variables that belong to the clause. The score is periodically divided by a constant.

There are also variants of the SAT problem. For instance, there is the MAX-SAT problem, an optimization variation of SAT, where we want to find a model to maximize the number of satisfied clauses. We can even weight the clauses differently, creating a

Weighted MAX-SAT problem. Moreover, we can generalize the SAT problem even further as a Satisfiability Modulo Theory (SMT) problem. An SMT is a satisfiability problem that allows descriptions using logical quantifiers, arithmetic operations, data structures such as arrays, lists, among others. Both DPLL and CDCL work as SMT solvers besides their original SAT origin. MAX-SAT also has its own generalization, called Optimization Modulo Theory (OMT).

3.2 Problem Encodings

In a work from 1992 [23], Henry Kautz and Bart Selman introduce the concept of planning as SAT, and develop a way to convert a planning problem into a SAT problem, by representing it as a set of axioms. The axioms determine the properties of the planning domain and instance, such as the property that, if the preconditions of an action hold, then the action achieves its effects, describe which propositions a given action does not affect when executed, and rule out anomalous models, such as the possibility of an action executing despite its preconditions being false, as well as axioms to guarantee that one, and only one, action occurs at a time. This initial problem encoding is referenced in future work as a *linear encoding*.

A later work by Kautz and Selman [26], published in 1996, discusses over three possible encodings for planning as satisfiability. Besides revisiting their linear encoding [23], they describe two additional problem encodings: one based upon GraphPlan [4] and a third state-based problem encoding. They describe the GraphPlan encoding as a direct conversion of a planning graph as defined in [4] into a boolean formula, as they can fully represent the graph through axioms that guarantee that:

- The initial state holds at layer 1, and the goals hold at the highest level;
- Each fact at level i implies the disjunction of all the operators at level $i - 1$ that have it as an add-effect;
- Operators imply their preconditions;
- Conflicting actions are mutually exclusive.

The advantage of this encoding is that it allows partially ordered plans, as the subgraph that serves as a solution to a planning is only partially-ordered, where the actions at a given level can be executed at any order given they are non-conflicting. This encoding is, however, less expressive than the linear encoding.

Finally, they [26] describe a third encoding called a state-based encoding. This encoding takes advantages from both previous encodings, being able to represent partially-ordered plans, as well as having a greater expressiveness if compared to the GraphPlan

encoding. The core of this encoding is defining axioms to ensure that each state is valid within itself, and axioms that represent the state transitions by expressing the possible actions that could account for the change *i.e.*the effects imply the action. Basically, the axioms define that, if a fluent changed its value between two time steps, then it means that an action that has this fluent in its effects (add or delete) must have been executed at that time step.

In this encoding the axioms that represent the planning instance can be classified into five groups of formulas, each representing a group of definitions and constraints. We consider the plan steps for this encodings, where, for a n-step plan, step 0 is the step before executing the first action, while step n is the step after executing the final action. We encode the initial state in a way that guarantees that every fact that belongs to that state holds at step 0 and every other fact does not.

$$\bigwedge_{f \in s_0} f_0 \wedge \bigwedge_{f \notin s_0} \neg f_0$$

We describe the goal state by the conjunction of all fluents (facts) that must hold at time step n. This states that all facts from the goal state must belong to the state one achieves by executing the final action in the plan.

$$\bigwedge_{f \in g} f_n$$

We state the fact that an action must be applicable in the current state if we want to execute it, *i.e.* its preconditions must hold at the current time step. We also state that the action's effects hold at the next time step, meaning that we modified the state by executing the action. In other words, any given action implies its preconditions and effects.

$$a_i \implies \left(\bigwedge_{p \in pre(a)} p_i \wedge \bigwedge_{e \in eff^+(a)} e_{i+1} \wedge \bigwedge_{e \in eff^-(a)} \neg e_{i+1} \right)$$

The axioms must also guarantee that an action *only* changes the facts that are in its effects. This means that an action can only modify what its definition describes. These are called *explanatory frame axioms*.

$$\left(\neg f_i \wedge f_{i+1} \implies \left(\bigvee_{a \in A | f_i \in eff^+(a)} a_i \right) \right) \wedge \left(f_i \wedge \neg f_{i+1} \implies \left(\bigvee_{a \in A | f_i \in eff^-(a)} a_i \right) \right)$$

Finally, we guarantee that only one action occurs at each time step, for each and every $a, b \in \mathcal{A}$. This is called a *complete exclusion axiom*. This axiom can be modified to only prevent the parallel execution of actions that are mutually exclusive, *i.e.* actions that have conflicting preconditions and/or effects.

$$\neg a_i \vee \neg b_i$$

This kind of formulation assumes a planning problem with a bounded horizon. This means that each set of formulas is meant to find a valid plan with a length (or horizon) n that reaches the goal state. This horizon is iteratively incremented until a valid model is found for the formula.

To illustrate the way the formulas above work, let's take the following example of an planning instance where a robot must move between different locations:

- Predicates:
  - $at(R, L)$, where R is a robot and L is a location.
- Actions:
  - $move(R, L1, L2)$, where R is a robot, L1 is a location, and L2 is another location. This action represents moving robot R from location L1 to location L2.
    - Preconditions: $at(R, L1)$. Robot R must be at location L1.
    - Effects: $not(at(R, L1)), at(R, L2)$. Robot R is no longer at location L1 and is now at location L2.
- Objects:
  - $r1 - robot$
  - $l1, l2 - locations$
- Initial state: $at(r1, l1)$. Robot r starts at location $l1$.
- Goal state: $at(r1, l2)$. We want robot $r1$ to be at location $l2$.

If we encode this example with the axioms presented above, with a planning horizon of 1, we get the following axioms:

- Initial state: $at(r1, l1, 0) \wedge \neg at(r1, l2, 0)$
- Goal state: $at(r1, l2, 1) \wedge \neg at(r1, l1, 1)$
- Actions:

  - $move(r1, l1, l2, 0) \Rightarrow at(r1, l1, 0) \wedge at(r1, l2, 1) \wedge \neg at(r1, l1, 1)$
  - $move(r1, l2, l1, 0) \Rightarrow at(r1, l2, 0) \wedge at(r1, l1, 1) \wedge \neg at(r1, l2, 1)$
- Explanatory frame axioms:
  - $\neg at(r1, l1, 0) \wedge at(r1, l1, 1) \Rightarrow move(r1, l2, l1, 0)$
  - $\neg at(r1, l2, 0) \wedge at(r1, l2, 1) \Rightarrow move(r1, l1, l2, 0)$
  - $at(r1, l1, 0) \wedge \neg at(r1, l1, 1) \Rightarrow move(r1, l1, l2, 0)$
  - $at(r1, l2, 0) \wedge \neg at(r1, l2, 1) \Rightarrow move(r1, l2, l1, 0)$
- Complete exclusion axiom: $\neg move(r1, l1, l2, 0) \vee \neg move(r1, l2, l1, 0)$

The resulting final formula is a conjunction of all the axioms defined above. Once the SAT solver finds a model for the formula with the given horizon, we then need to extract the plan from the model. We know that, in a scenario with no parallel actions, for each time step i from 1 to n, there will be only one ground action fluent with a truth-value of *true*. That fluent represents the $i - th$ action of the plan.

To address the issue of the resulting large problem encodings, which tends to increase the solving time, the authors [26] discuss that a way to mitigate that issue is to reduce the predicate arity, so if there is, for instance, a quaternary predicate `move(x,y,z,i)` that represents that an object `x` moves from `y` to `z` at time `i`, we can instead represent it as 3 binary predicates `object(x,i)`, `source(y,i)`, and `dest(z,i)`, which reduces the size of the encoding. Having the encoding defined, they generate SAT instances for bounded increasing planning horizons until a solution is found. The boolean formula is then formed from all the axioms and can be solved with a regular SAT solver, where any found model that satisfies the formula is a valid plan.

In a later work by Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler, published in 1997 [12], the authors develop a new encoding that relaxes the representation even more. In this encoding, they take plan parallelism even further, allowing parallel actions given that the actions can be laid in some order where they do not conflict with each order, *i.e.* there is, at least, one valid ordering for the actions. This differs from GraphPlan parallelism in [26], where every possible ordering of the actions needs to be valid. There is a further relaxation of this technique, described in [54], where a group of actions does not need to have all of its preconditions satisfied to be applied in parallel, given that the actions can satisfy themselves using each other's effects in some ordering.

3.3 SAT Solving for Planning as SAT

The work by Kautz and Selman [23, 26] uses a SAT solving approach that solves SAT-encoded planning problems sequentially, one at a time, for bounded increasing hori-

zon lengths (1, 2, 3, 4, 5, ...). Work by Rintanen *et al.* [48] introduces two parallel algorithms, which we call algorithms A and B. Algorithm A simultaneously solves SAT instances for bounded horizons of size 1 through n. If it finds that the formula is satisfiable, it returns the plan, and the algorithm terminates. Otherwise, it starts a solver for the shortest plan length not searched yet, always solving n SAT instances in parallel. Algorithm B also runs a predefined number of SAT solvers in parallel, but the CPU time it assigns to solver with horizon t is the multiplication of the time assigned to the instance with horizon $t-1$ and a constant $g < 1$, decreasing the CPU time as the horizon grows. It also restricts the number of parallel SAT solvers depending on memory availability.

Since some approaches use the CDCL algorithm for SAT solving [54, 47], there is research on planning-specific heuristics to replace standard heuristics, such as VSIDS. A recent one is the Variable Selection to Satisfy Goals and Subgoals heuristic [46]. This heuristic is based on the principle that each goal literal has to be made true by an action, and that for that action to be executed, its preconditions must be made true by some previous action, or be true at the initial state. It then performs a backwards search to find the earliest time point where a goal literal becomes true and remains true to the end of the plan. It then chooses the action that makes that literal true.

3.4 SAT-Based Planners

Kautz and Selman [23] developed the SATPLAN [23, 26, 22] system, evolving it as they found better encodings. They further refined it, leveraging on the state-based encoding [26]. Kautz and Selman also developed the BLACKBOX planner [24], which works by creating a GraphPlan-style [4] planning graph with a bounded horizon, translating the constraints into a set of clauses, solving it with a regular SAT solver and, if a solution is found, translating the solution into a plan and pruning unnecessary actions. If a solution is not found, it increases the horizon and tries again. This planner obtained great results in the 1998 AIPS planning competition [32].

The current state-of-the-art SAT planner is Jussi Rintanen's Madagascar planner [47]. This planner leverages on the parallel actions encoding described in [54] and the GraphPlan binary mutexes. Furthermore, the authors describe more than one version for the planner, with different configurations, which allows the user to experiment with different execution settings and use the most efficient one for their application. The versions vary the horizon length the planner considers when searching for plans with bounded horizons, as well as the heuristic to use with CDCL algorithm. The authors cite two heuristics currently implemented in the planner: the VSIDS heuristic [38] and the Variable Selection to Satisfy Goals and Subgoals heuristic [46].

4. MULTIAGENT PLANNING

The formalism we discussed so far assume there is either a single agent acting in the environment, or at least a centralized decision-making agent generating plans. However, in many cases, there is not only one agent, but a group of agents that must cooperate to complete a given task. In that case, the planner must take that into account, and find the most efficient set of actions for a group of agents to execute and reach the common goal. This is called a multiagent planning (MAP) task. Figure 2.1 depicts an example of a multiagent planning environment with two robot agents and multiple possible goals, where we might want to derive a plan to direct our robots to one of the goals, where they shall cooperate to achieve it, or direct each robot to a different goal. In this Chapter, we summarize background for multiagent planning, defining the formalism and notation in Section 4.1, the language used to describe a multiagent planning problem, in Section 4.2, and some frameworks and algorithms, in Section 4.3.

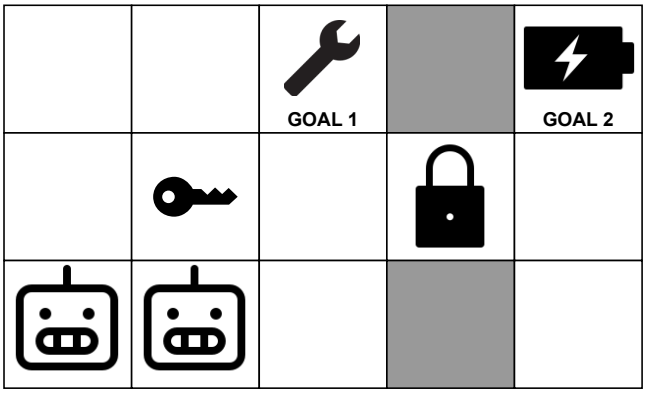

Figure 4.1 – Multiagent planning environment example

4.1 Formalism and Notation

For convenience, we use the definitions from [52]. To define a MAP task, we use the MA-STRIPS definition [7], which extends the previously discussed STRIPS definition to multiagent tasks, as this is the main formalism adopted by the works in this field. MAP shares most definitions with the standard STRIPS definition, such as the definition of *states*, *facts*, and *actions*, as well as the definitions of preconditions, effects, and state applicability for actions.

We define a MAP task as a tuple $T_{MAP} = \langle \Phi, \Sigma, \{\mathcal{A}^i\}_{i=1}^{n}, \mathcal{I}, G \rangle$, where:

- Φ is a finite set of k agents;
- Σ is a finite set of facts;

- $\mathcal{A}^i$ is the finite set of actions for agent i. We define the full set of actions for T_{MAP} as $\mathcal{A}$, where $\mathcal{A} = \cup_{\forall i \in \Phi} \mathcal{A}^i$;
- $\mathcal{I}$ is the initial state;
- G is the common goal state.

The solution for a MAP task is an ordered sequence of actions, *i.e.*a plan, that, when applied to initial state $\mathcal{I}$, generates a state that contains the goal state G. In the standard MA-STRIPS definition, a solution plan must be a totally-ordered set of actions [7], while other works describe a plan as a set of action sequences (one sequence per agent) [30] or as a partially-ordered sequences of actions [53].

We can divide the facts Σ in a MAP task into *public* and *private* facts. Private facts are internal to a given agent $i \in \Phi$, and can only be used and affected by actions in $\mathcal{A}^i$. Public facts are accessible to all agents. We denote agent's i private (internal) facts as Σ^i_{int} and the public facts as Σ_{pub}. We distribute the task to the agents in Φ as a set of *local views*, where each agent has its own local view of the task. The local view for agent i of the MAP task T is denoted as a 4-tuple $T_i = \langle \Sigma^i, \mathcal{A}^i, \mathcal{I}^i, G \rangle$, where:

- $\Sigma^i = \Sigma^i_{int} \cup \Sigma_{pub}$ is the set of facts accessible to agent i;
- $\mathcal{A}^i$ are the available actions for i;
- $\mathcal{I}^i \in \Sigma^i$ is the set of facts belonging to the initial state $\mathcal{I}$ that are accessible to i;
- G is the common goal state. All facts in the common goal state are accessible to all agents in Φ.

4.2 Problem Description Language

Just as single-agent planning has its standard language for defining domains and tasks (PDDL), so does MAP. In MAP, the most widely used language for defining a task is Multiagent PDDL (MA-PDDL) [28, 29]. MA-PDDL is very similar to standard PDDL, being nothing but a multiagent extension to standard PDDL. We discuss the differences below.

In MA-PDDL, we can define tasks with both *factored* and *unfactored* privacy. Unfactored privacy tasks have a domain definition file and a single task file that serves all the agents. Factored tasks have a domain definition file and an individual task file for each agent, representing the agent's local view. This is done by adding the `:factored-privacy` or `:unfactored-privacy` property to the `requirements` section. Listing 4.1 depicts an example of a domain definition in a factored definition [52].

This example represents the *Transport Agent* domain. In this domain, there are transport agencies that work in different geographical areas and transport packages from locations within their areas. Additionally, there is also a factory that resides in the intersection of the agencies area and needs raw material to manufacture a product. The transport agencies must transport the packages of raw material from their current location to the factory, so that the factory can use it to manufacture the final product. A transport agency can only transport a package to the factory if the initial package location resides within its area.

As we can see, the domain definition is fairly similar to a single-agent PDDL domain file. The main difference we see is in the `:factored-privacy` key word, which we have already discussed, and the `:private` key word within the `:predicates` section. The `:private` key word states that the predicates within that scope will not be shared across the agents, meaning that each transport agency will not disclose any information neither about the topology of their work area nor about their trucks.

```
(define (domain transport-agency)
  (:requirements
    :factored-privacy
    :typing
    :equality
    :fluents
  )
  (:types
    transport-agency
    area
    location
    package
    product - object
    truck
    place - location
    factory - place
  )
  (:predicates
    (manufactured ?p - product)
    (at ?p - package ?l - location)
    (:private
      (area ?ag - transport-agency ?a - area)
      (in-area ?p - place ?a - area)
      (owner ?a - transport-agency ?t - truck)
      (pos ?t - truck ?l - location)
      (link ?p1 - place ?p2 - place)
    )
  )
  (:action drive
    :parameters (?ag - transport-agency ?a - area ?t - truck ?p1 - place
 ?p2 - place)
```

```
      :precondition (and
                      (area ?ag ?a)
                      (in-area ?p1 ?a)
                      (in-area ?p2 ?a)
                      (owner ?a ?t)
                      (pos ?t ?p1)
                      (link ?p1 ?p2)
                    )
      :effect (and
                (not (pos ?t ?p1))
                (pos ?t ?p2)
              )
    )
    [...]
  )
```

Listing 4.1 – Excerpt from *Transport Agency* domain described in MA-PDDL

Having the domain defined, we can define the task itself. In this example, since it is a factored definition, we have a task file for each agent. Listing 4.2 shows the MA-PDDL task file for agent 1. Firstly, we see that it is not so different from a single-agent problem definition. Secondly, we see that only facts regarding the transport agency 1 are described in the initial state, as the ones regarding any other transport agencies are private to those agencies. Finally, we see that the goal is to have the final product manufactured, as the agencies must work together to make sure that the factory has the necessary raw material to do so. We assume that the factory is another agent and that it will manufacture the product if the transport agencies provide the raw material.

```
  (define (problem ta1)
    (:domain transport-agency)
    (:objects
      ta1 - transport-agency
      ga1 - area
      l1 l2 sf - place
      p - package
      fp - product
    )
    (:init
      (area ta1 ga1)
      (pos t1 l1)
      (owner t1 ta1)
      (at p l1)
      (link l1 l2)
      (link l2 l1)
      (link l1 sf)
      (link sf l1)
      (link l2 sf)
```

```
20        (link sf l2)
21        (in-area l1 ga1)
22        (in-area l2 ga1)
23        (in-area sf ga1)
24      )
25      (:goal (manufactured fp))
26    )
```

Listing 4.2 – Excerpt from *Transport Agency* domain described in MA-PDDL

4.3 Frameworks and Algorithms

Since this survey is related to SAT-based techniques, an important planner to mention is μ-SATPLAN [11], which first distributes the goals among the agents and then iteratively feeds each agent with the solution of the previous agent as input. Each agent solves the task using the regular SATPLAN [22] planner, and the agents progressively solve the entire task. Still, this work does not bring anything new into the SAT planning field itself, as it just breaks the task down into single-agent ones and then uses the regular single-agent SATPLAN planner to solve them.

As we see from the approach above, as well as from other approaches, it is common in multiagent planning to just translate a problem into a single-agent one, or to break it down into multiple single-agent ones, to leverage from common single-agent planners. Some examples of such approaches are Multi-Agent Planner by Plan Reuse (MAPR) [6], Plan Merge by Reuse (PMR) [33], and Centralized Multi-Agent Planning (CMAP) [13]. MAPR has a similar approach to μ-SATPLAN, where it distributes the goals to the agents and each agent receives as input, the output of the previous agent. The difference is that each agent uses the LAMA [45] planner to solve its task. PMR also uses the LAMA planner in as similar way as MAPR, but instead of making the agents solve the task sequentially, it parallelizes the agents, making each of them generate a plan for the distributed goals, and then merges the plan in post-processing. Finally, CMAP also uses LAMA, but it completely translates the MAP task into a standard single-agent planning task, and then uses LAMA to solve it.

5. GOAL RECOGNITION

Goal recognition is the task of correctly identifying an agent's goal by observing its interactions with the surrounding environment [51]. Such observations may translate into actions performed by the agent or properties of the environment during the agent's actions. We define a goal recognition problem over planning domain theory in the same manner as [43, 44] defined a plan recognition problem. Figure 5.1 illustrates an example of a goal recognition environment. In this example, we observe the agent's movements, depicted in blue, and since we can see that the agent is moving towards the wrench, we could infer that the wrench is the agent's goal. In this Chapter, we outline the formalism and notation, in Section 5.1, and discuss some frameworks for solving goal recognition problems, in Section 5.2

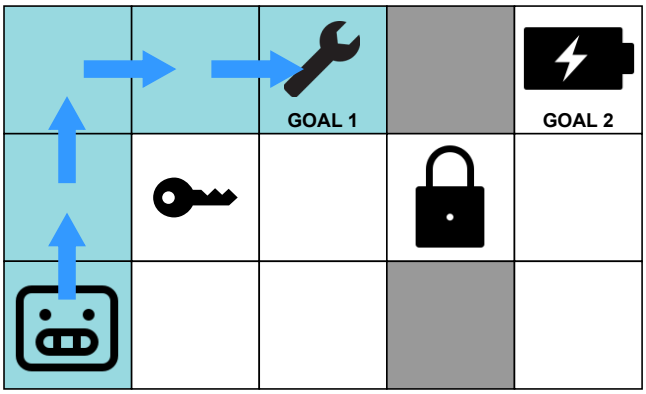

Figure 5.1 – Goal recognition environment example

5.1 Formalism and Notation

Goal recognition is often mentioned as plan recognition, a variant where one desires to not only recognize the agent's goal, but also the plan this agent is executing to achieve it and that explains the observations. The earlier approaches to plan recognition [25, 18] were based on plan libraries, where the recognizer would assume that the agent was following a plan taken from a previously defined plan library. The algorithms were dependent of this plan library, and the frameworks would generate the plans before the recognition process started. The recognizer would then try to find which plan from the library explained the observations. This was the common approach until a paper by Ramirez and Geffner [43] was published, where the authors interpreted a plan recognition problem as a problem over a domain theory, or a planning domain theory, using the formalism and techniques of automated planning as a base to formulate and solve plan recognition problems. Their work paved the way to many other works, and shaped the

state-of-the-art of how we tackle plan and goal recognition problems. For this reason, this is the kind of approach we focus on this survey. Analogously, the majority of goal and plan recognition solutions that are based on planning domain theory use the PDDL language [36] as a base to define the problems.

A goal recognition problem over domain theory can be defined as a tuple $T_{GR} = \langle \Xi, \mathcal{I}, \mathcal{G}, O \rangle$, in which $\Xi = \langle \Sigma, \mathcal{A} \rangle$ is a planning domain definition; $\mathcal{I}$ is the initial state; $\mathcal{G}$ is the set of goal state hypotheses, which includes the correct intended goal state G^* (*i.e.*, $G^* \in \mathcal{G}$); and $O = \langle o_1, o_2, ..., o_n \rangle$ is an observation sequence of executed actions, where each observation $o_i \in \mathcal{A}$.

The solution to a goal recognition problem is the hidden goal G^* achieved by the sequence of observations O. We can have either full or partial observations, meaning that we may be observing all the actions performed by the agent or only a subset (contiguous or not) of such actions. Furthermore, observations may contain noise within them, meaning that the observation set may include fake or spurious actions, that were not actually performed by the agent. In real-world applications, faulty sensors may cause noise in observations, for instance.

5.2 Frameworks and Algorithms

When we deal with single-agent goal recognition, we see no relevant work on SAT-based approaches. Given that, we will focus this Section on discussing other relevant goal recognition frameworks. We do not focus on approaches based on plan libraries, as they have been largely surpassed by the ones based on planning domain theory.

In [43], the authors develop the formalism to treat a goal recognition problem as a problem over domain theory, instead of using a previously generated plan library and looking for a plan included in the library, which can be costly and unfeasible in many applications. In one of the introduced approaches, the authors use a planner to obtain the cost to each goal, using that cost as a maximum when reusing the planner with the observations added to the goal state, checking if there is an optimal plan that can reach the goal through the observations. As mentioned before, this work is one of the most relevant works in the field, as it changed the way we see plan recognition problems.

Later work by the authors [44] introduces a way to discover the real goal by calculating a probability distribution over the set of goal hypotheses. In this later work, the authors compute the probability of each goal hypothesis given the observations. To do that, they use a Bayesian model, and use the cost difference between plans that follow the observations versus plans that deviate from the observations. To generate those plans, they use off-the-shelf classical planners, such as HSP [5] and LAMA [45].

A more recent approach [42] is based on landmark analysis. In this work, the authors develop two heuristics to analyze the landmarks for each goal hypothesis, and compare the achieved landmarks to the total landmarks for each of them. The first heuristic computes the ratio between achieved and total landmarks to provide a score for each goal hypothesis. The second heuristic computes the same ratio, but assigns a higher value to landmarks that are landmarks to fewer goal hypotheses, since achieved landmarks that are not shared across different goals provide more information on the agent’s intention.

6. MULTIAGENT GOAL RECOGNITION

In Chapter 4 we discussed over another prism of planning problems where one might want to plan for multiple agents to cooperate as a team in pursuit of a given goal. Conversely, we also have interest in identifying when multiple agents are cooperating as a team, and what goal they are pursuing. This derives a subgroup of goal recognition called *multiagent goal recognition* (MAGR).

Figure 6.1 illustrates an example of a multiagent goal recognition environment. In this example, we observe the both agents' movements, depicted in blue and green. Since we can see that the blue agent is moving towards the wrench, while the green one is moving towards the battery, we could infer that each agent is it's own team, and that the wrench is the blue agent's goal, while the battery is the green agent's goal. In this Chapter, we describe the formalism and notation for MAGR in Section 6.1, as well as some algorithms and techniques, in Section 6.2.

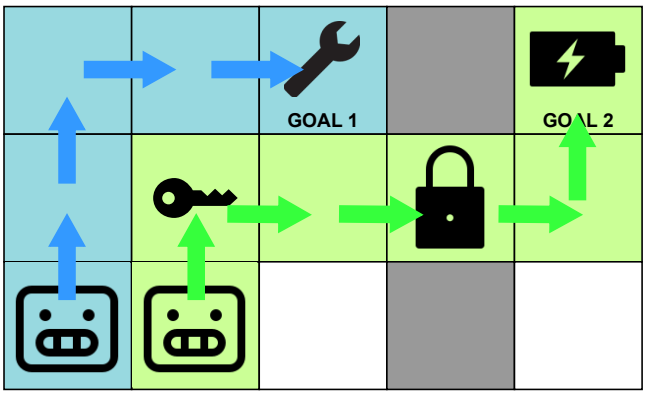

Figure 6.1 – Multiagent goal recognition environment example

6.1 Formalism and Notation

To define a MAGR problem, we extended the definition of a single-agent goal recognition problem. The main difference is that we have a set of agents, instead of a single one. This alone adds another variable to the problem, as we have to analyze which actions were executed by which agent. Additionally, the agents may cooperate on teams, where each team is pursuing a different goal. Our objective is to find out how are these teams organized, *i.e.* how many teams are there, who (which agents) are the members of each team, and which goal each team is pursuing. We call this a team-goal mapping.

The different works in MAGR diverge in how they define a task. They sure have a common ground, such as defining the actions, with names, preconditions and effects, defining the possible facts and the initial state, defining the list of agents involved in the

process, and defining the observations, or team traces. What the MAGR papers we discuss here mostly differ is in how they treat the possible goals being pursued. For instance, the work by Zhuo *et al.* [55, 56, 57] clearly define a set of goal hypotheses for the framework to analyze, while in the work by Argenta and Doyle [1, 2], the authors define the set of hypotheses as the set of all possible goals, not defining a finite set specifically. For convenience, we will define a MAGR task as close as to the definition we gave to a single-agent one, with the appropriate adaptations taken from the papers we discuss here.

We define a MAGR task as a tuple $T_{MAGR} = \langle \Xi, \Phi, \mathcal{I}, \mathcal{G}, \mathcal{O} \rangle$, where Ξ is the domain definition, with the possible facts and actions; $\Phi = \phi_{i=1}^{k}$ is the set of agents, where $k > 0$ and where each agent is a part of one, and only one team, where a team is a group of one or more agents; $\mathcal{I}$ is the initial state; $\mathcal{G}$, where $|\mathcal{G}| = l$, is the set of goal hypotheses, where $l \geq k$, meaning there is the possibility that each agent is its own team pursuing its own goal. Each goal hypothesis might or might not be a correct intended goal for one of the teams, and every correct intended goal for each of the teams is included in this list. We denote the hidden subset of correct intended team goals as G^*; $\mathcal{O} = \{\mathcal{O}_i\}_{i=1}^{k}$ are the team traces, which are a list of observed actions mapped to the agent who executed them. The solution to a MAGR task is the correct identification of the teams, *i.e.* how many teams are there and who are the members in each team, and the correct intended goal for each team.

6.2 Frameworks and Algorithms

In this Section, we discuss relevant works in the field of multiagent goal and plan recognition. We divide this Section into the different kind of approaches. In Section 6.2.1, we discuss a classic "plan recognition as planning" approach based on the work by Ramirez and Geffner [43]. In Section 6.2.2, we discuss probabilistic approaches for multiagent goal and plan recognition. Finally, in Section 6.2.3, we discuss SAT-based approaches for goal and plan recognition, as it is the focus of this survey.

6.2.1 A Classical "Plan Recognition as Planning" Approach

In a 2016 work by Argenta and Doyle [1], the authors develop a framework called Multiagent Plan Recognition as Planning (MAPRAP) inspired by the one developed by Ramirez and Geffner [43], using a planner to compute the cost to achieve each goal and then comparing that cost to the cost of a plan that incorporates the observations. If the plan cost increases with the observations, then the observations do not explain that goal, and hence that is not the correct intended one. They describe two approaches,

MAPRAP^A and *MAPRAP*^{B}. *MAPRAP*^{A} maps all possible teams to all possible goals to the hypotheses and tests the cost for each one of them, removing when the observations increase the cost. *MAPRAP*^{B}, on the other hand, starts with a single team, containing all agents, for each goal, and prunes the agents from the composition when their actions increase the cost. This makes so that *MAPRAP*^{B} reduces the number of runs per goal as the agent/team ratio increases.

6.2.2 Probabilistic Approaches

Following the work above, the same authors extend the MAPRAP framework in a probabilistic way to not only return the goals and plans identified as the possible correct intended ones, but to rank them using a likelihood score [2]. They call this new framework Probabilistic Multiagent Plan Recognition as Planning (P-MAPRAP). They do this by computing a likelihood score for each possible interpretation (an interpretation maps every agent to a team and every team to a goal) and storing them in a priority queue based on said score. Before reading the first observable, they compute a baseline score for each interpretation in the list without considering observations. They iterate over the list of observables computing the cost with the observations, just like MAPRAP. They compute the likelihood score using the cost difference, and store the interpretations back in the priority queue, repositioning inside the queue according to the score. If the new top interpretation does not include the current observations, they remove it from the queue and repeat the process until it does. This guarantees that the framework only considers the most likely interpretations.

In a different work [49], the authors propose three probabilistic approaches to multiagent plan recognition problems. In this work, they transform the goal recognition problems into temporal planning problems, *i.e.* problems where each action has a defined temporal duration in the environment. The first approach is based on Ramirez and Geffner's probabilistic plan recognition work [44], using the cost difference between a plan with and without the observations as a proxy to compute the probability of the observations given each goal, and then using this probability to compute the probability of each goal hypothesis being the correct one given the observations.

The second approach differs from the first by not using this cost difference to compute the probabilities, and by using a diverse planner, as in [50]. In a diverse planning problem, the objective is to obtain a set of m plans that are, at least, at a distance of d away from each other [50]. In this approach, the authors compute the probability of each plan generated by the diverse planner being the followed one given the observations, and use this probability as a proxy to compute the probability of each goal hypothesis being the correct intended one given the observations. One interesting feature to note is

that during the probability computation, they add penalties for unexplained and missing observations in the plans. The third and final approach, the one that achieved the best results, is a hybrid one between the first two, where they use a diverse planner, but merge the generated plans.

6.2.3 SAT-Based Approaches

The work by Zhuo *et al.* [55, 56, 57] is one of the greatest references in the field. The reason it correlates to our survey is the fact that all of these approaches work by building a set of constraints from the planning problem and solve them using a weighted MAX-SAT solver. The main difference between them is that [55] uses team plan libraries as inputs, and [57] works with incomplete team traces and action-models, which would be analogous to working with incomplete domain models [41]. Since we are focusing on goal recognition over a domain theory instead of plan libraries, and we are not focusing on goal recognition with incomplete domain models, we focus on discussing [56].

In [56], the authors develop a framework that, given a partially-observed team trace, a set of action models (defined as STRIPS actions), an initial state, and a set of goal hypotheses, outputs a set of team plans with the maximum likelihood to achieve some goal among the hypotheses. They use what they call a likelihood function to compute the likelihood of a set of team plans achieving a goal. The set of team plans must respect a series of properties, which they ensure by building sets of constraints.

First, they build a set of candidate activities by instantiating the actions using the observations and the initial and goal states. Then, they build a set of hard constraints to ensure some conditions, that the set of team plans is a partition of the trace, covers all the observed activities, and turns the initial state into a goal state. With the hard constraints generated, they generate a set of soft constraints based on the likelihood function. Finally, they group all the constraints and solve them using a MAX-SAT solver. They do that for each goal hypothesis, and the framework considers the one with the highest result from the MAX-SAT solver to be the correct intended one. The framework then converts the found solution for the MAS-SAT problem into a plan and returned as well. The results are promising, but the authors have limited the evaluation to simpler problem domains.

7. CONCLUSION

In this survey, we have reviewed the theoretical background, formalism, and notations for both single-agent and multiagent planning and goal recognition. We have also reviewed the theoretical background around the Boolean Satisfiability Problem (SAT) and SAT-based techniques for planning. Finally, we have revisited relevant work on these areas, including the few SAT-based goal recognition ones.

We saw that while SAT is widely used in planning applications, it is not that widely used in goal recognition applications. The lack of experimentation with SAT-based goal recognition in more complex domains, while we see that kind of experimentation with other techniques, may be an indication that these SAT-based solutions would not scale well in more complex settings, and so the majority of research effort has been directed to other approaches. We still expect for more experimentation on SAT-based goal recognition techniques, as there might be more to explore in this field that has not yet been explored.

REFERENCES

[1] Argenta, C.; Doyle, J. “Multi-agent plan recognition as planning (maprap)”. In: Proceedings of the 8th International Conference on Agents and Artificial Intelligence, 2016, pp. 141–148.

[2] Argenta, C.; Doyle, J. “Probabilistic multi-agent plan recognition as planning (p-maprap): Recognizing teams, goals, and plans from action sequences”, 2017, pp. 575–582.

[3] Bacchus, F. “Aips 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems”, *Ai Magazine*, vol. 22, 2001, pp. 47–56.

[4] Blum, A. L.; Furst, M. L. “Fast planning through planning graph analysis”, *ARTIFICIAL INTELLIGENCE*, vol. 90, 1995, pp. 1636–1642.

[5] Bonet, B.; Geffner, H. “Hsp: Planning as heuristic search”, *Planning Competition at Artificial Intelligence Planning Systems 1998 (AIPS’98)*, 1998.

[6] Borrajo, D. “Multi-agent planning by plan reuse”. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, 2013, pp. 1141–1142.

[7] Brafman, R.; Domshlak, C. “From one to many: Planning for loosely coupled multi-agent systems”. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling, 2008.

[8] Cook, S. A. “The complexity of theorem-proving procedures”. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[9] Davis, M.; Logemann, G.; Loveland, D. “A machine program for theorem-proving”, *Communications of the ACM*, vol. 5, 1962, pp. 394–397.

[10] Davis, M.; Putnam, H. “A computing procedure for quantification theory”, *J. ACM*, vol. 7–3, jul 1960, pp. 201–215.

[11] Dimopoulos, Y.; Hashmi, M. A.; Moraitis, P. “μ-satplan: Multi-agent planning as satisfiability”, *Knowledge-Based Systems*, vol. 29, 2012, pp. 54–62, artificial Intelligence 2010.

[12] Dimopoulos, Y.; Nebel, B.; Koehler, J. “Encoding planning problems in nonmonotonic logic programs”. In: Proceedings of the Fourth European Conference on Planning, 1997, pp. 169–181.

[13] Fernández, S.; Borrajo, D. “Mapr and cmap”. In: Proceedings of the Competition of Distributed and Multi-Agent Planners, 2015, pp. 1–3.

[14] Fikes, R. E.; Nilsson, N. J. “STRIPS: A new approach to the application of theorem proving to problem solving”, *Journal of Artificial Intelligence Research (JAIR)*, vol. 2–3, 1971, pp. 189–208.

[15] Geib, C. W. “Problems with Intent Recognition for Elder Care”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2002, pp. 13–17.

[16] Geib, C. W.; Goldman, R. P. “Plan Recognition in Intrusion Detection Systems”. In: DARPA Information Survivability Conference and Exposition (DISCEX), 2001.

[17] Ghallab, M.; Nau, D. S.; Traverso, P. “Automated Planning - Theory and Practice.” Elsevier, 2004.

[18] Goldman, R. P.; Geib, C. W.; Miller, C. A. “A new model of plan recognition”. In: Conference on Uncertainty in Artificial Intelligence, 1999.

[19] Helmert, M. “The fast downward planning system”, *Journal of Artificial Intelligence Research (JAIR)*, vol. 26–1, jul 2006, pp. 191–246.

[20] Hoffmann, J. “Ff: The fast-forward planning system”, *AI Magazine*, vol. 22–3, Sep 2001, pp. 57.

[21] Karpas, E.; Magazzeni, D. “Automated planning for robotics”, *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3–1, 2020, pp. 417–439, https://doi.org/10.1146/annurev-control-082619-100135.

[22] Kautz, H. “Deconstructing planning as satisfiability.” In: Proceedings of The Twenty-First National Conference on Artificial Intelligence, 2006.

[23] Kautz, H.; Selman, B. “Planning as satisfiability”. In: Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI’92), 1992, pp. 359–363.

[24] Kautz, H.; Selman, B. “Blackbox: A new approach to the application of theorem proving to problem solving”, *AIPS98 Workshop on Planning as Combinatorial Search*, 06 1998.

[25] Kautz, H. A.; Allen, J. F. “Generalized plan recognition”. In: AAAI Conference on Artificial Intelligence, 1986.

[26] Kautz, H. A.; Selman, B. “Pushing the envelope: Planning, propositional logic and stochastic search”. In: AAAI/IAAI, Vol. 2, 1996.

[27] Kelly, J.-P.; Botea, A.; Koenig, S. “Offline planning with hierarchical task networks in video games”, *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 4–1, Sep 2021, pp. 60–65.

[28] Komenda, A.; Štolba, M.; Kovács, D. “The international competition of distributed and multiagent planners (codmap)”, *AI Magazine*, vol. 37, 10 2015, pp. 109–115.

[29] Kovacs, D. L. “Complete bnf definition of ma-pddl with privacy”. Source: http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF-20150221.pdf, 2015.

[30] Kvarnström, J. “Planning for loosely coupled agents using partial order forward-chaining”, *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 21–1, Mar 2011, pp. 138–145.

[31] Levin, L. A. “Universal sequential search problems”, *Problems of Information Transmission*, vol. 9–3, 1973.

[32] Long, D.; Kautz, H.; Selman, B.; Bonet, B.; Geffner, H.; Koehler, J.; Brenner, M.; Hoffmann, J.; Rittinger, F.; Anderson, C. R.; Weld, D. S.; Smith, D. E.; Fox, M.; Long, D. “The aips-98 planning competition”, *AI Magazine*, vol. 21–2, Jun 1998.

[33] Luis, N.; Fernández, S.; Borrajo, D. “Plan merging by reuse for multi-agent planning”. In: Proceedings of the 2nd ICAPS Workshop on Distributed and Multi-Agent Planning, 2014, pp. 38–44.

[34] Marques Silva, J.; Sakallah, K. “Grasp-a new search algorithm for satisfiability”. In: Proceedings of International Conference on Computer Aided Design, 1996, pp. 220–227.

[35] Marques-Silva, J.; Sakallah, K. “Grasp: a search algorithm for propositional satisfiability”, *IEEE Transactions on Computers*, vol. 48–5, 1999, pp. 506–521.

[36] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; Wilkins, D. “PDDL — The Planning Domain Definition Language”, *The Fourth International Conference on Artificial Intelligence Planning Systems 1998 (AIPS’98)*, 1998.

[37] Mirsky, R.; Gal, Y. K.; Shieber, S. M. “CRADLE: An Online Plan Recognition Algorithm for Exploratory Domains”, *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8–3, 2017, pp. 45:1–45:22.

[38] Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; Malik, S. “Chaff: engineering an efficient sat solver”. In: Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), 2001, pp. 530–535.

[39] Oh, J.; Meneguzzi, F.; Sycara, K. “Probabilistic plan recognition for proactive assistant agents”. In: *Plan, Activity, and Intent Recognition: Theory and Practice*, Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; Bui, H. H. (Editors), Elsevier, 2014, pp. 275–288.

[40] Oh, J.; Meneguzzi, F.; Sycara, K.; Norman, T. J. “Prognostic normative reasoning”, *Engineering Applications of Artificial Intelligence*, vol. 26–2, 2013, pp. 863 – 872.

[41] Pereira, R.; Meneguzzi, F. “Goal recognition in incomplete domain models”. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2018.

[42] Pereira, R. F.; Oren, N.; Meneguzzi, F. “Landmark-Based Heuristics for Goal Recognition”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2017.

[43] Ramírez, M.; Geffner, H. “Plan Recognition as Planning”. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009.

[44] Ramírez, M.; Geffner, H. “Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners”. In: Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI), 2010.

[45] Richter, S.; Westphal, M. “The lama planner: Guiding cost-based anytime planning with landmarks”, *Journal of Artificial Intelligence Research*, vol. 39–1, sep 2010, pp. 127–177.

[46] Rintanen, J. “Heuristics for planning with sat”. In: Principles and Practice of Constraint Programming – CP 2010, Cohen, D. (Editor), 2010, pp. 414–428.

[47] Rintanen, J. “Madagascar: Scalable planning with sat”, 2014.

[48] Rintanen, J.; Heljanko, K.; Niemelä, I. “Planning as satisfiability: parallel plans and algorithms for plan search”, *Artificial Intelligence*, vol. 170–12, 2006, pp. 1031–1080.

[49] Shvo, M.; Sohrabi, S.; McIlraith, S. A. “An ai planning-based approach to the multi-agent plan recognition problem (preliminary report)”. In: Proceedings of the The AAAI 2017 Workshop on Plan, Activity, and Intent Recognition (PAIR), 2017.

[50] Sohrabi, S.; Riabov, A. V.; Udrea, O. “Plan recognition as planning revisited”. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016, pp. 3258–3264.

[51] Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; Bui, H. H. “Plan, Activity, and Intent Recognition: Theory and Practice”. Elsevier, 2014.

[52] Torreño, A.; Onaindia, E.; Komenda, A.; Štolba, M. “Cooperative multi-agent planning: A survey”, *ACM Comput. Surv.*, vol. 50–6, nov 2017.

[53] Torreño, A.; Onaindia, E.; Sapena, O. “An approach to multi-agent planning with incomplete information”, *Frontiers in Artificial Intelligence and Applications*, vol. 242, aug 2012.

[54] Wehrle, M.; Rintanen, J. “Planning as satisfiability with relaxed e-step plans.” In: AI 2007 : Advances in Artificial Intelligence: 20th Australian Joint Conference on Artificial Intelligence, Surfers Paradise, Gold Coast, Australia, December 2-6, 2007, Proceedings, 2007, pp. 244–253.

[55] Zhuo, H.; Li, L. “Multi-agent plan recognition with partial team traces and plan libraries”. In: IJCAI, 2011.

[56] Zhuo, H.; Yang, Q.; Kambhampati, S. “Action-model based multi-agent plan recognition”. In: Advances in Neural Information Processing Systems, Pereira, F.; Burges, C.; Bottou, L.; Weinberger, K. (Editors), 2012.

[57] Zhuo, H. H. “Recognizing multi-agent plans when action models and team plans are both incomplete”, *ACM Trans. Intell. Syst. Technol.*, vol. 10–3, may 2019.