AIP Conference Proceedings

RESEARCH ARTICLE | SEPTEMBER 06 2017

A hybrid bit-encoding for SAT planning based on clique-partitioning FREE

Cristóbal Tapia; Pablo San Segundo; Ramón Galán

AIP Conf. Proc. 1872, 020015 (2017)
https://doi.org/10.1063/1.4996672

29 May 2024 11:54:47

Boost Your Optics and Photonics Measurements
Lock-in Amplifier
Zurich Instruments
Find out more
Boxcar Averager

AIP Publishing

A Hybrid Bit-encoding for SAT Planning Based on Clique-Partitioning

Cristóbal Tapia[1,a)] , Pablo San Segundo[1] and Ramón Galán [2]

[1] *ETSIDI, Universidad Politécnica de Madrid,Spain*
[2]*ETSII, Universidad Politécnica de Madrid,Spain*

[a)] correspondence author: cristobal.tapia@upm.es

29 May 2024 11:54:47

Abstract. Planning as satisfiability is one of the most efficient ways to solve classic automated planning problems. In SAT planning, the encoding used to convert the problem to a SAT formula is critical for the performance of the SAT solver. This paper presents a novel bit-encoding that reduces the number of bits required to represent actions in a SAT-based automated planning problem. To obtain such encoding we first build a conflict graph, which represents incompatibilities of pairs of actions, and bitwise encode the subsets of actions determined by a clique partition. This reduces the number of Boolean variables and clauses of the SAT encoding, while preserving the possibility of parallel execution of compatible (non-neighbor) actions. The article also describes an appropriate algorithm for selecting the clique partition for this application and compares the new encodings obtained over some standard planning problems.

INTRODUCTION

One of the most efficient algorithms for automated planning in classical environments is planning as satisfiability (also SAT planning). This approach is based on reducing a planning problem to Boolean satisfiability (SAT) and finds its justification in the upsurge of very fast SAT solvers in the last decade. An important caveat about this reduction to propositional logic is how to keep the explosion of logical variables and clauses within reasonable bounds. This paper addresses this problem and proposes a technique based on clique partitioning that reduces the number of logic variables of the SAT instance. The paper is structured as follows: First we introduce the basic definitions and notation concerning PDDL and SAT planning concepts, bit encoding and clique partitioning. We then present the technique for SAT encoding based on a clique model. The final part of the paper presents empirical validation and describes future work and improvements.

CLASSICAL AUTOMATED PLANNING AS BOOLEAN SATISFIABILITY

Automated Planning is a part of Artificial Intelligence that states the problem of selecting a course of actions to reach a goal. Classical automatic planning employs a planning model that on a domain description — typically an initial state and a goal state— and a set of actions that can change the current state. Classical domains have some restrictions: they are fully observable, deterministic, finite, static (that is, changes occur only when the planning agent acts), and discrete (in time, action, objects, and effects). Thus, the planning problem can be clearly defined and solved using a logical approach [1, 2]. Formally this is defined as a planning task $P = (V, A, s_0, s_g)$ where:

Mathematical Methods and Computational Techniques in Science and Engineering
AIP Conf. Proc. 1872, 020015-1–020015-8; doi: 10.1063/1.4996672
Published by AIP Publishing. 978-0-7354-1552-2/$30.00

- $V = \{v_1, v_2, \dots v_n\}$ is a set of finite domain state variables.
- A is a set of actions, which consist in a pair (pre_a, eff_a) of partial variable assignments, called preconditions and effects respectively.
- S_0 is a complete variable assignment denoted *initial state*
- S_g is a partial variable assignment, denoted *goal state*

A plan is a sequence of actions that lead from an initial state to a goal state. An action a can be applied on a state S_n if pre_a is true in S_n. The application of a changes the current state of the planning agent S_n to a new one, S_{n+1}, according to its effects eff_a. The solution to a planning problem is a sequence of actions that, starting from the initial state, transfers the world of the planning agent to the goal state. Planning has PSPACE complexity [3].In the paper, we use the Planning Domain Definition Language (PDDL) to define planning problems. To the best of our knowledge, PDDL the most extended language for automated planning [4, 5, 6, 7].

Planning as Satisfiability (also denoted SAT Planning) is a way to solve planning problems by translating a planning formalism (such as STRIPS or PDDL) into a sequence of SAT problem instances [8, 9]. In this approach, the planning problem is compiled into a CNF (conjunctive normal form) Boolean expression, where logical variables refer to domain entities such as facts (predicates) and actions. The reduction to SAT is such, that the action variables assigned a truth value in any solution to SAT are also a feasible solution to the planning problem. For this to be possible, the SAT reduction requires an a priori fixed time horizon, which is expressed as a number of steps (also denoted periods).

Although classical planning allows the execution of exactly one action per time step, it is very interesting to allow for concurrent actions, providing these are not conflicting in the same step. The alternative is to add more time steps, and thus more logical variables and clauses in the SAT encoding. Moreover, plans which contain concurrent non-conflicting actions are expected to be shorter.

The reduction of the planning problem to a SAT instance can be done using different techniques. In this paper — and based on mainstream encoding schemes [10]— all predicate instances are grounded and a Boolean variable is assigned to each of them. To denote the state in which the predicate occurs, and also record predicate changes, a consecutive step number is attached to the name of each predicate. Actions are also grounded and encoded as logic variables in a similar way. The process can generate an explosion of logical variables and clauses, so a number of techniques have been described to narrow the number of variables [11, 12, 13, 14, 15, 16, 17].

As an example, `MOVE(robot1,room1,room2)` represents a PDDL grounded action, `ROBOT-AT(robot1, room1)` represents a PDDL grounded predicate, `move_robot1_room1_room2@1` is a logic variable that references to the move grounded action, applied on time step 1, and `robot-at_robot1_room1@3` is another logic variable that refers to the grounded predicate `ROBOT-AT`, and occurs on time step 3.

When translating actions to SAT, there are some alternative encodings:

- **Regular** - each grounded action is encoded as a different logical variable. This produces as many logical variables as grounded actions (and may be exponential).
- **Simple Operator Splitting** - each grounded action is encoded with as many logical variables as parameters (including the step parameter). This way, the same logical variable can be used on different grounded actions.
- **Overloaded Operator Splitting** - this is a generalization of Simple Operator Splitting, in which all actions can use the same logical variable for the same parameter, no matter if they are different actions.
- **Bitwise** - each grounded action is encoded as a number, expressed as a binary value, and the logical variables encode the bits needed for the binary value. This encoding requires the minimum number logical variables.
- **Hybrid** - a combination of different encodings.

In general, we are interested in the most compact representation provided by the bitwise encoding. The downside is that it cannot handle concurrent actions, as the same variables are used to encode the binary value of all the possible actions in the domain. To sidestep this problem, hybrid encodings have been proposed in literature: some of the grounded actions or predicates are encoded bitwise, and the rest regular. This is a somewhat more complex solution which leads to other problems.

29 May 2024 11:54:47

Motivated by the above considerations, we propose *a bitwise encoding over sets of actions that are incompatible between them*. We preprocess the original PDDL problem to detect incompatibilities between actions, and then group together actions that cannot be applied at the same time, so they can be safely encoded bitwise. This approach allows parallelism on compatible actions and, at the same time, takes advantage of the compact bit representation. Notice the encoding to SAT must be done after grounding all PDDL actions and predicates. Prior to grounding, it is useful to detect *static* (also called *invariant*) predicates. Static predicates are those that will not change during plan execution, because no action adds or removes them upon execution. These can be eliminated from the planning problem before translating it to SAT.

Table 1. Types of relations between predicates and their compatibility.

Type	Relation between predicates	Compatibility	Boolean Expression
Between preconditions			
I	Same	Compatible	$\text{pre}(A_1) \cap \text{pre}(A_2) \neq \emptyset$
II	Opposite	Mutual Exclude	$\text{pre}(A_1) \cap \neg\text{pre}(A_2) \neq \emptyset$
Effects			
III	Same	*Race condition*	$\text{add}(A_1) \cap \text{add}(A_2) \neq \emptyset$ $\text{del}(A_1) \cap \text{del}(A_2) \neq \emptyset$
IV	Opposite	Inconsistency	$\text{add}(A_1) \cap \text{del}(A_2) \neq \emptyset$ $\text{del}(A_1) \cap \text{add}(A_2) \neq \emptyset$
Preconditions and effects			
V	Same	*Overwrite*	$\text{pre}(A_1) \cap \text{add}(A_2) \neq \emptyset$ $\neg\text{pre}(A_1) \cap \text{del}(A_2) \neq \emptyset$
VI	Opposite	Interference	$\text{pre}(A_1) \cap \text{del}(A_2) \neq \emptyset$ $\neg\text{pre}(A_1) \cap \text{add}(A_2) \neq \emptyset$

There are some types of incompatibilities between actions in the same step that can be detected by preprocessing the PDDL actions when grounded. Table 1 shows a categorization of these types of incompatibilities, based on relations between the preconditions and effects of actions. The column header *Relation between predicates* denotes whether the predicates are both negated or both not negated, or one is negated and not the other. The column *Boolean Expression* contains the logic formula related to each type. A_1 and A_2 represent two different actions, *pre* refers to action preconditions (negated or not), and *add*, *del* denote the action effects of addition and removal respectively. To note, negated preconditions are optional in PDDL (and specified by the *negative-preconditions* requirement). A conversion between domains with and without this requirement is always possible, so our analysis preserves generality.

A BIT ENCODING BASED ON INCOMPATIBLE ACTIONS

We propose an algorithm to partition the set of all grounded actions for a particular step, such that each subset contains pairwise incompatible actions, suitable for a bitwise representation in the equivalent SAT problem. To compute the partition, we first build a *conflict* (also *incompatibility*) *graph* $G_c = (V, E)$ such that:

1. Each vertex represents a ground action.
2. Two vertices $u, v \in V$ are connected by an edge iff the corresponding actions are incompatible in the same step of the plan.

We then process the graph in the following way:

29 May 2024 11:54:47

1. Identify a clique partition $A = \{A_1, A_2, \dots, A_n\}$ of V, such that it minimizes the number of bits needed for the encoding of the planning problem (B). The total number of bits for any subset is:

$$B = \sum_i ceil(\log_2 |A_i + 1|$$

2. Encode each subset A_i, $(1 \le i \le n)$, with $ceil\ (\log_2|A_i + 1|)$ fresh logical variables. Note that we need one more bit than $|A_i|$ for all subsets, since it is also necessary to consider the case where none of the ground actions are selected in a step of the plan.

Clearly, since the edges in the graph represent incompatible actions, any clique in the graph is a mutually exclusive set of actions which can therefore be bit encoded preserving concurrency in the plan. When using this approach, each action in A_i, $(1 \le i \le n)$, will be encoded as a Boolean expression of a set of logic variables, instead of only one dedicated variable. The advantage is that it is not necessary to include mutual exclusion clauses between actions belonging to the same subset. On the other hand, it is necessary to add new logical expressions to deal with inconsistent combinations of values of the variables in A_i.

Graph partitioning using cliques

As shown in the previous section, we are interested in finding a clique partition of the vertices of a conflict graph G_c which represents incompatibilities between pairs of ground actions in a plan. We describe in this section a heuristic algorithm to efficiently compute the partition.

A *clique* in $G,$ is any induced subgraph of G such that all its vertices are pairwise adjacent. The largest clique in G is called the maximum clique and its size, $\omega(G)$, is the *clique number* of the graph. Finding the largest clique in the graph is known as the *maximum clique problem*, a well known and deeply studied NP-hard problem from graph theory. The key intuition, mentioned in the previous subsection, is that any clique in the conflict graph G_c determines a set of actions which are all pairwise incompatible in the same step. Thus, we are interested in finding a clique partition $K = \{K_1, K_2, \dots K_l\}$ such that $\bigcup_{i=1}^{l} K_i = V$, $\bigcap_{i=1}^{l} K_i = \phi$ and each S_i subset is as large as possible

Possibly the most efficient way to compute a clique partition is to use a coloring heuristic over the complement graph $\overline{G_c}$ [18]. However, we note that not all the cliques are equally interesting. In particular, since we are using a bit representation for the actions, k bits can at most encode $2^k - 1$ actions (with the null value representing the absence of action). For example, a clique (subsets of actions) of size 5 and a clique of size 7 both use 3 bits in our compact bitwise representation, and cliques of size 8 and 15 both use 4 bits. For this reason, we need a heuristic which gives as much control as possible of the size of the cliques during execution. We propose in this paper an algorithm based on *iteratively finding, and removing, a subset of large disjoint cliques at every step*, which is better suited for this specific problem.

Moreover, the idea of iterative clique extraction has also been described in literature for clique partitioning. In [19], a clique heuristic is used to determine at each step a set of non-disjoint cliques S, and then again to determine a subset $S_d \subseteq S$ of disjoint cliques which will be a subset of the desired clique partition. At each step, the set of nodes in S_d are removed from the conflict graph and the procedure continues until the clique partition is complete.

To get the maximum possible control over the size of the cliques that make up the partition, we propose a modified version of the previous algorithm, which, at each step, uses an efficient exact maximum clique algorithm to compute a pool of (overlapping) maximum cliques. The size of this pool is limited by a threshold T and all maximum cliques over that number are discarded. The subset of disjoint cliques from the pool is obtained by building a new disjoint graph G_d, where the nodes are all the cliques from the pool and two edges (cliques) are connected if they are disjoint (i.e., they share no common nodes). Solving the maximum clique problem exactly for G_d gives as all possible disjoint cliques candidates S_d that may belong to a subset of n cliques of the desired partition K. Let $K_i = \{C_1, C_2, \dots, C_m\}$, be the chosen m disjoint cliques at the ith iteration, where $K_i \subseteq S_d$, and $|S_d| = n$ $(n \ge m)$. These m cliques are the ones that best fit the compromise between the total number of bits required for the encoding of K_i, given as $\sum_{i=1}^{n} ceil\ (\log_2(|C_i| + 1|))$, and the *degree of saturation* d of disjoint clique C_i, defined as $d(C_i) = 2^{(\log_2(|C_i|+1|)} - |C_i|$. The total saturation for, K_i , is $d(K_i) = \sum_{i=1}^{n} d(C_i)$.

29 May 2024 11:54:47

Table 2. Outline of the algorithm BCLQ.

Algorithm BCLQ
INPUT: An action conflict graph $G_c = (V, E)$, a threshold T
OUTPUT: A clique partition K of G_c

1. **Repeat until** $V = \phi$
2. Compute the set of all maximum cliques (up to T) and store them in S
3. Build the disjoint graph G_d related to S
4. Compute the set of all maximum cliques in G_d (up to T) and store them in S_d
5. Add the best combination of cliques in S_d to K
6. Remove all nodes in s from V

Algorithm BCLQ in Table 2 outlines the main steps of the procedure. To compute the exact maximum clique for both graphs, G_c and G_d, we use the efficient bit-parallel algorithm BBMCX [20][21], tailored to output all maximum cliques and not just one of them. We note that although the maximum clique problem is NP-hard, the conflict graphs generated from typical planning problems are solvable, in practice, for several thousands of ground actions in a few seconds. Domains with high incompatibility between actions may give rise to an exponential number of solutions of the conflict graph. A good compromise between efficiency and the reduction in the representation of the actions is achieved when threshold T is set to 5000.

AN EXAMPLE

The following example presents a simple planning problem in PDD and describes the grounding of actions and how to build the incompatibility graph. The planning domain may have many (in this case 2) robots that can move from between two rooms if there is a path between them. Every robot can pick up to one box on one room, carry it, and drop it in a different room. The constraints are the following:

1. If a robot picks a box, no other robot can take it at the same time.
2. A robot cannot move in-between rooms twice in the same step.

We use PDDL STRIPS and typing requirements to describe the planning problem. Typing reduces the number of grounded actions and is cleaner and more compact.. The PDDL domain contains these actions:

```
(:action move :parameters ( ?t - robot ?x - room ?y - room )
    :precondition ( and ( ROBOT-AT ?t ?x )
                        ( PATH ?x ?y )    )
    :effect (and (ROBOT-AT ?t ?y)
                 (not (ROBOT-AT ?t ?x)) )                 )
(:action load :parameters (?t - robot ?r - room ?b - box)
    :precondition (and  (BOX-AT ?b ?r )
                        (ROBOT-AT ?t ?r )
                        ( not (ROBOT-LOADED ?t) )
    :effect (and (ROBOT-LOADED ?t)
                 (not (BOX-AT ?b ?r ))
                 (ROBOT-CARRIES ?t ?b )    )              )
```

The initial and goal states are defined as follows:

```
(:objects   bot1 bot2 - robot
            rm1 rm2 - room
            bx1 bx2 - box)
(:init
    (ROBOT-AT bot1 rm1) (ROBOT-AT bot2 rm1)
    (BOX-AT bx1 rm1) (BOX-AT bx2 rm2)
    (PATH rm1 rm2)  (PATH rm2 rm1)  )
(:goal
    (and (ROBOT-AT bot1 rm2) (ROBOT-AT bot2 rm2 )
         (BOX-AT bx1 rm2) (BOX-AT bx2 rm2)       )  )
```

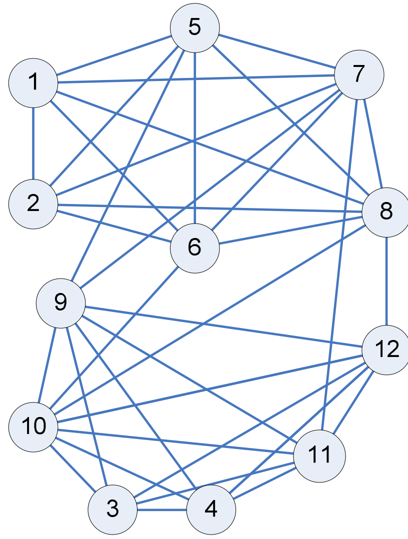
29 May 2024 11:54:47

The grounded actions in any step appear in Table 3. To note, the grounding algorithm takes into account static predicates. In the example, the `PATH` predicate (that describes the links between rooms) has been previously detected as static, and therefore actions such as `move_bot1_rm1_rm1`, do not appear in the table —note that it there is no path from a room to itself.

Table 3. A set of ground actions.

1	**move_bot1_rm1_rm2**	**5**	**load_bot1_rm1_bx1**	**9**	**load_bot2_rm1_bx1**
2	**move_bot1_rm2_rm1**	**6**	**load_bot1_rm1_bx2**	**10**	**load_bot2_rm1_bx2**
3	**move_bot2_rm1_rm2**	**7**	**load_bot1_rm2_bx1**	**11**	**load_bot2_rm2_bx1**
4	**move_bot2_rm2_rm1**	**8**	**load_bot1_rm2_bx2**	**12**	**load_bot2_rm2_bx2**

With respect to action compatibility, *action #1* has predicate `¬bot1_rm1` as part an effect, whereas *action 2* contains `bot1_rm1`. This is a conflict so both actions are incompatible, and so are *actions #3* and *#4*. A different type of incompatibility occurs between *actions #1* and *#5*. Here, assertion of the predicate `ROBOT-AT_rm1` by one action invalidates a precondition predicate of the other. *Actions #5* and *#6* are incompatible because of the same reason, but this time the predicate `ROBOT-LOADED` is the cause. Note this predicate has been added to the domain to prevent a robot carrying two boxes at the same time. Finally, *actions #5* and *#9* are also conflicting, this time because of the conflict over the `BOX-AT_bx1_rm` predicate.



(a)

#	Action	x_1	x_2	x_3
0	**no action**	**0**	**0**	**0**
1	**move_bot1_rm1_rm2**	**0**	**0**	**1**
2	**move_bot1_rm2_rm1**	**0**	**1**	**0**
3	**load_bot1_rm1_bx1**	**0**	**1**	**1**
4	**load_bot1_rm1_bx2**	**1**	**0**	**0**
5	**load_bot1_rm2_bx1**	**1**	**0**	**1**
6	**load_bot1_rm2_bx2**	**1**	**1**	**0**
7	**not used**	**1**	**1**	**1**

(b)

Figure 1. Conflict graph $\boldsymbol{G_c}$ for the actions (a). The bit encoding of a clique of actions in $\boldsymbol{G}$ (b)

The incompatibility graph G_c for the example is depicted in Figure 1 (a). The two cliques of incompatible actions obtained when applying the proposed BCLQ algorithm over the example graph are $K_1 = \{1, 2, 5, 6, 7, 8\}$ and $K_2 = \{3, 4, 9, 10, 11, 12\}$. Subset K_1 has cardinality 6 so we need at least 3 bits to encode each different bit combination. In (b) we give a sample encoding for K_1 using Boolean variables x_1, x_2, x_3. Another 3 variables would be necessary to encode K_2.

We show the way to use this bit representation by a simple example: instead of using a Boolean variable for the ground action `load_bot1_rm2_bx2` (*action #6* in Table 3), we now use the Boolean expression $x_1 \wedge \overline{x_2} \wedge x_3$. The rest of actions in the domain would require a similar treatment.

29 May 2024 11:54:47

RESULTS

Table 4 reports the reduction in the number of logical variables obtained for a SAT encoding of the actions of a number of planning problems. The first and second problems are based on robot actions domain, they are similar to the example explained with more actions. The other problems are based on PDDL domains, commonly used on tests and planning competitions. Note that the encoding still allows for concurrent actions, which would not be the case in a standard bitwise representation. The table provides the number of nodes n (grounded actions), the number of bits needed for the encoding with the proposed method (*bits*), the number of cliques found (*groups*) and the number of nodes of any maximum clique in the partition (*max)*. Finally, header *reduction* measures the reduction in the number of bits with respect to the bits required by a regular encoding.

Table 4. Bit encodings of some planning problems. *p* is the average density of the graph. Header *Groups* refer to the size of the clique partition, *max* is the maximum size of any clique in the partition, *bits* is the total number of variables required to encode the actions and *reduction* measures the compression obtained as a percentage wrt to a regular encoding.

	Graph			**Set Partition**			
Name	n	m	p	**groups**	**max**	**bits**	**reduction**
robot_C_Pm01	544	14528	0,037	52	72	174	68.0%
robot_C_Pg02	2508	163536	0,015	190	216	593	76.4%
aipslog01	405	5682	0,071	68	14	182	55.1%
aipslog04	618	8577	0,072	100	11	292	52.7%
blocksworld01	441	57771	0,007	19	86	70	84.1%
blocksworld02	891	203139	0,004	31	132	108	87.9%
hanoi-4	126	5286	0,023	7	37	27	78.6%
hanoi-8	572	79892	0,007	11	101	58	89.9%
slidetile 1	192	5840	0,032	7	40	38	80.2%
slidetile 3	192	5840	0,032	7	40	38	80.2%

The results show a significant compression, which would even be more relevant over problems with many pairs of incompatible actions. Some planning domains don't allow many parallel actions, due the way they are defined. In these cases the benefits of applying some action in parallel are not so significant and such problems will have longer plans, so reducing the number of variables can have a big impact on the SAT solving time. With respect to performance, the proposed BCLQ algorithm is very fast, and the CPU time spent on all these examples is less than one second. This is not significant with respect to full SAT compilation process.

CONCLUSIONS

We have presented an efficient heuristic that aims to minimize the number of bits required to encode the actions of a planning problem as Boolean satisfiability, while preserving concurrency of actions. The heuristic first builds a conflict graph of grounded actions and finds a clique partition. Each clique subset represents a set of mutually incompatible actions, which can be bit encoded safely without losing concurrency. To build the clique partition a state-of-the art maximum clique algorithm is used, which, at each step, extracts the largest clique from the graph. The cliques thus obtained will overlap, so a final step selects from these the desired partition. This work is part of a planning and simulation system[22].

In the examples reported, the new method reduces by close to a 90% the amount of logical variables required to represent the ground actions in CNF-SAT when using a regular encoding. Future work includes using this technique as a metric of the number of possible concurrent actions in a planning domain. For example, it could be used in portfolio-based algorithm selection, since some planning solvers perform better when concurrency is high.

29 May 2024 11:54:47

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (grant NAVEGASE: DPI2014-53525-C3).

REFERENCES

1. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice, Morgan Kaufmann Publishers (2004)
2. Russell, S. J., Norvig, P.: Artificial Intelligence: A Modern Approach.3rd Ed. Prentice Hall (2009)
3. Bylander, T., 1994 The computational complexity of propositional STRIPS planning. *Artificial Intelligence,* 69(1-2).pp.165–204
4. McDermott, D. et al. 1998, PDDL -- The Planning Domain Definition Language -- Version 1.2.
5. Fox, M., & Long, D., 2003, PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, 20, pp. 61-124
6. Helmert, M., 2008, Changes in PDDL 3.1.
7. Linares, C. Jiménez, S., García-Olaya, A., 2015, The deterministic part of the seventh International Planning Competition, *Artificial Intelligence*, Volume 223, June 2015, pp. 82-119
8. Kautz, H. A., Selman, B., 1992, Planning as Satisfiability (SAT - Plan). *Proceedings of the 10th European Conference on Artificial Inteligence*, 9.
9. Kautz, H. A., Selman, B., 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. *In Proc. 13th Nat. Conf. on AI*, pp. 1194-1201.
10. Ernst, M.D., Millstein, T.D. & Weld, D.S., 1997. Automatic SAT-compilation of planning problems. *IJCAI International Joint Conference on Artificial Intelligence*. pp. 1169–1176.
11. Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. . *IJCAI International Joint Conference on Artificial Intelligence*, pp. 318–325.
12. Robinson, N. et al., 2007. A Compact and Efficient SAT Encoding for Planning. *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*.
13. Wehrle, M. & Rintanen, J., 2007. Planning as Satisfiability with relaxed \exists-step plans. *Proceedings of 20th Australasian Joint Conference on Artificial Intelligence (AI-07),* Gold Coast, Australia. pp. 244–253.
14. Helmert, M., 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6), pp.503–535.
15. Huang, R., Chen, Y. & Zhang, W., 2010. A Novel Transition Based Encoding Scheme for Planning as Satisfiability. *AAAI Conference on Artificial Intelligence*, pp.89–94.
16. Rintanen, J. 2011. Madagascar: efficient planning with sat. *The 2011 International Planning Competition*, pp. 61.
17. Sideris, A., & Dimopoulos, Y. 2010, Constraint Propagation in Propositional Planning. *ICAPS,* pp. 153-160.
18. Jin, Y., Hao. J.K., Hybrid evolutionary search for the minimum sum coloring problem of graphs, *Information Sciences*, Elsevier, vol. 352-353. 2016, pp. 15-34.
19. Wu, Q., Hao, J.K., Improved Lower Bounds for Sum Coloring via Clique Decomposition, 2013, arXiv:1303.6761.
20. San Segundo, P., Nikolaev, A., Batsyn, M., 2015. Infra-chromatic bound for exact maximum clique search, *Computers and Operations Research*, vol. 64, C. Elsevier, pp. 293-303.
21. San Segundo, P., Nikolaev, A., Batsyn, M., Pardalos, P. M., 2016. Improved Infra-Chromatic Bound for Exact Maximum Clique Search. *Applied Intelligence*, vol. 45 issue 3 pp. 463-487.
22. Tapia, C., San Segundo, P., & Artieda, J., 2015. A PDDL-Bassed Simulation System. Proceedings of the IADIS International Conference Intelligent Systems and Agents.

29 May 2024 11:54:47