Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability

Enrico Giunchiglia
DIST — Università di Genova
Viale Causa 13
16145 Genova, Italy

Alessandro Massarotto
DIST — Università di Genova
Viale Causa 13
16145 Genova, Italy

Roberto Sebastiani
IRST, 38050 Povo,
Trento, Italy

Abstract

In this paper we focus on Planning as Satisfiability (SAT). We build from the simple consideration that the values of fluents at a certain time point *derive deterministically* from the initial situation and the sequence of actions performed till that point. Thus, the choice of actions to perform is the only source of nondeterminism. This is a rather trivial consideration, but which has important positive consequences if implemented in current planners via SAT. In fact, it produces a dramatic size reduction of the space of the truth assignments searched in by the SAT decider used to solve the final SAT problem. To justify this claim, we repeat many of the experiments reported in (Ernst, Millstein, & Weld 1997), and show that the CPU time requested to solve a problem can go down up to 4 orders of magnitude.

Introduction

Historically, planning has been dealt with as a problem of deduction in first order logic (Green 1969; McCarthy & Hayes 1969) or later through the definition of special purpose algorithms (see e.g. (Blum & Furst 1995)). Kautz and Selman (1992) proposed a different approach, where any planning problem is first limited in size (by looking for plans whose length is $\leq n$ for some fixed n), and then reduced to a satisfiability problem in propositional logic. Several reductions are possible, each having a different asymptotic size and number of variables (see (Kautz & Selman 1992; 1996; Kautz, McAllester, & Selman 1997; Ernst, Millstein, & Weld 1997)).

In this paper we build from the following simple consideration:

> The values of fluents at a certain time point *derive deterministically* from the initial situation and the sequence of actions performed till that point. Thus, the choice of actions to perform is the only source of nondeterminism.

This is a rather trivial consideration, but which has important positive consequences if implemented in current planners via SAT. In fact, it produces a dramatic size reduction of the space of the truth assignments searched in by the SAT decider used to solve the final SAT problem. To experimentally support this claim we consider many of the problems reported in (Ernst, Millstein, & Weld 1997) and distributed with the MEDIC planner.[1] We generate various SAT encodings with MEDIC and solve each of them using both the TABLEAU system[2] –as done in (Ernst, Millstein, & Weld 1997)– and TABLEAU*, a version of TABLEAU modified in order to incorporate the above idea. Finally, we compare the results and show that:

- The search space effectively searched by TABLEAU* (i.e. the number of nodes in the search tree) is up to 4 orders of magnitude less than the search space analyzed by TABLEAU.
- The CPU time requested by TABLEAU* to solve a problem is up to 4 orders of magnitude less than the CPU time requested by TABLEAU on the same problem.

Planning as Satisfiability

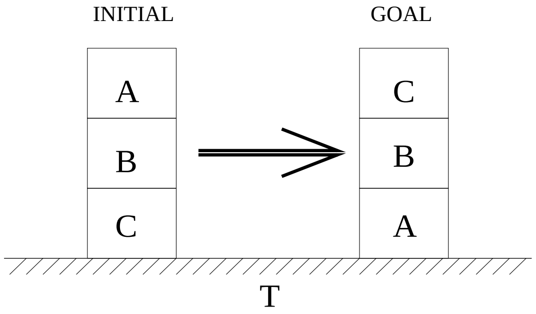
In the following, we will restrict our attention to six of the eight automatic encodings which have been studied and comparatively experimentally evaluated in (Ernst, Millstein, & Weld 1997). More in detail, we consider the

- explanatory-regular-parallel-eliminate (erpe),
- classical-regular-sequential-eliminate (crse),
- explanatory-simple split-sequential-eliminate (ecse),
- classical-simple split-sequential-eliminate (ccse),
- explanatory-bitwise-sequential-eliminate (ebse),
- classical-bitwise-sequential-eliminate (cbse).

It is not our goal to describe these six encodings in detail. This has been already done in (Ernst, Millstein,

[1] The MEDIC system is available at http://www.cs.washington.edu/research/projects/ai/-www/planning.html.

[2] The TABLEAU system is available at http://www.cirl.uoregon.edu/crawford/crawford.html.



$Move(b, s, d)$

PRECOND : $Block(b) \wedge Clear(b) \wedge On(b, s) \wedge$
$(Clear(d) \vee Table(d)) \wedge$
$b \neq s \wedge b \neq d \wedge s \neq d$

EFFECT : $Clear(s) \wedge \neg On(b, s) \wedge$
$On(b, d) \wedge \neg Clear(d)$

Figure 1: A Blocks World, Initial and Goal States

& Weld 1997). We will recall the underlying ideas and illustrate them through the simple blocks reverse problem in Figure 1. This planning problem is characterized by the single operator *Move*, 4 individuals (the 3 blocks A, B, C and the table T), initial and goal states as in the Figure.

We will use the same notation and terminology employed in (Ernst, Millstein, & Weld 1997). Thus, time takes nonnegative integer values, fluents occur at even time points, and actions occur at odd time points. We represent the fluent e.g. $On(A, B)$ at time t by $On_t(A, B)$, and the action e.g. $Move(A, B, C)$ at time t by $Move^r_t(A, B, C)$. Notice the index r to actions: it indicates it is a representation which varies according to the particular encoding used.

Before illustrating the various encodings, we recall that, for any given planning problem and plan length $n \geq 0$, the clauses corresponding to the following facts always belong to the final SAT problem:

- For any action, if it is executed at time $0 < t < 2n$ then its preconditions and its effects must be satisfied at time $t-1$ and at time $t+1$ respectively. For example, for the action $Move^r_t(A, B, C)$ we will have (*action axioms*)

$$\begin{aligned} &Move^r_t(A, B, C) \supset \\ &\quad Clear_{t-1}(A) \wedge On_{t-1}(A, B) \wedge Clear_{t-1}(C) \wedge \\ &\quad Clear_{t+1}(B) \wedge \neg On_{t+1}(A, B) \wedge \\ &\quad On_{t+1}(A, C) \wedge \neg Clear_{t+1}(C). \end{aligned}$$

- At time 0 the formula specifying the initial state must be satisfied. In our example, all the fluents are assumed to be false at time 0 except for

$$On_0(A, B), On_0(B, C), On_0(C, T), Clear_0(A).$$

- At time $2n$ the formula specifying the goal states must be satisfied:

$$On_{2n}(C, B) \wedge On_{2n}(B, A) \wedge On_{2n}(A, T).$$

Finally, we always employ the optimizations described in (Ernst, Millstein, & Weld 1997) and implemented in the MEDIC planner, such as factoring and type optimizations. These allow to reduce the number of actions (e.g. the actions in which b is not a block, or in which $b = s$ or $b = d$ or $s = d$ are not generated), the number of fluents (e.g. fluents like $Table(A)$ are automatically considered to be false at each even time step), and the size of the encodings.

The encodings differ for the "Frame axioms representation" (Classical/Explanatory), "Action representation" (Regular/Simple-split/Bitwise), "Planning strategy" (Sequential/Parallel).

Frame axioms representation

Classical (McCarthy & Hayes 1969): any action specifies that the non affected fluents keep their values. For example, moving A from B to T does not affect whether block C is clear or not:

$$Move^r_t(A, B, T) \wedge Clear_{t-1}(C) \supset Clear_{t+1}(C),$$
$$Move^r_t(A, B, T) \wedge \neg Clear_{t-1}(C) \supset \neg Clear_{t+1}(C).$$

"At-least-one-action" axioms guarantee that at each odd time step some action occurs, thus preventing fluents from changing freely due to non-action:

$$\bigvee_{\substack{b,s,d \in \{A,B,C,T\} \\ b \neq s, b \neq d, s \neq d, b \neq T}} Move^r_t(b, s, d).$$

Explanatory (Haas 1987): If a fluent changes its value, some action affecting it must have occurred. For example, if block C becomes unclear, then either we have moved A or B on top of C:

$$\begin{aligned} &\neg Clear_{t+1}(C) \wedge Clear_{t-1}(C) \supset \\ &\quad Move^r_t(A, B, C) \vee Move^r_t(A, T, C) \vee \\ &\quad Move^r_t(B, A, C) \vee Move^r_t(B, T, C). \end{aligned}$$

Action representation

Regular : each action is represented by a distinct variable, e.g. $Move^r_t(A, B, C)$ is $Move_t(A, B, C)$. In our example there are 18 actions and thus we will have 18 variables per odd time step.

Simple-split : each n-ary action is replaced by the conjunction of n distinct variables, e.g. $Move^r_t(A, B, C)$ is

$$MoveArg1_t(A) \wedge MoveArg2_t(B) \wedge MoveArg3_t(C).$$

In our example we will have 11 variables per odd time step.

Bitwise : each action corresponds to a conjunction of $\lceil log_2 \mathcal{A} \rceil$ literals, where $\mathcal{A}$ is the number of actions. The conjunction corresponds to a sequence of bits numbering the set of action. In our example 5 bits are sufficient. If we number the set of actions according to the lexicographic order, $Move^r_t(A, B, C)$ is

$$\neg Bit1_t \wedge \neg Bit2_t \wedge \neg Bit3_t \wedge \neg Bit4_t \wedge \neg Bit5_t.$$

Planning Strategy

Sequential : for each pair of actions α and β, add axioms of the form $\neg\alpha_t^r \vee \neg\beta_t^r$ for each odd time step. For example, we will have:

$$\neg Move_t^r(A,B,C) \vee \neg Move_t^r(A,B,T).$$

These axioms ensure that at each time step at most one action is executed. They are not included (since not necessary) for encodings using a bitwise action representation (i.e. ebse and cbse) or the classical frame axioms (i.e. crse, ccse and cbse).

Parallel : for each pair of actions α and β, add axioms of the form $\neg\alpha_t^r \vee \neg\beta_t^r$ for each odd time step if α effects contradict β preconditions. For example, we will have

$$\neg Move_t^r(B,T,A) \vee \neg Move_t^r(A,B,C).$$

These axioms do not prohibit to execute more than one action at some time step, and the result is a plan whose actions form a partial order. However, imposing any total order compatible with the partial order constitutes a valid plan.

Act, and the rest will follow

In the next two sections, we will use the symbol L to denote a literal; the symbol A_t^i to denote the i-th action variable at time t; the symbol F_t^j to denote the j-th fluent variable at time t; the symbols $\mathbf{A}_t$ and $\mathbf{F}_t$ to denote respectively the sets of action variables $\{A_t^1, A_t^2, \ldots\}$ and fluent variables $\{F_t^1, F_t^2, \ldots\}$ at time t; the expressions *preconds*(A_t^i) and *effects*(A_t^i) to denote the sets of precondition and effect variables respectively of action A_t^i. We denote with $\mathcal{A}$ and $\mathcal{F}$ the cardinality of $\mathbf{A}_t$ and $\mathbf{F}_t$ respectively (which do not depend on t), and with n the length of the searched plan.

The work underlying this paper starts from observing that, independently from the encoding used, the following simple facts hold:

1. for each t, the truth values of the fluent variables in $\mathbf{F}_t$ derive deterministically from the initial values $\mathbf{F}_0$ and from the truth values of the action variables in $\mathbf{A}_1, \ldots, \mathbf{A}_{t-1}$, representing all the actions performed before t. Thus:
2. the choice of the truth values for the action variables A_t^i's is the only source of non-determinism in SAT-encoded planning.

These facts translate into SAT-encoded planning the consideration about determinism in the introductory section.

Consider, for instance, the regular representation of actions and the classical frame axioms. For each odd time t, if all the fluent variables F_{t-1}^j have been assigned a truth value and one action variable A_t^i has been assigned true, then all the fluent variables F_{t+1}^j which are in *effects*(A_t^i) can be deterministically assigned as a consequence of the action axioms, while all the other fluent variables F_{t+1}^k can be assigned as a consequence of the frame axioms. Notice that the reverse is not true, that is, the values of actions A_t^i's are not a deterministic consequence of the values of the variables in $\mathbf{F}_{t-1}$ and in $\mathbf{F}_{t+1}$.

Current SAT-based planners do not take advantage of these issues. In fact, the SAT decider run over the SAT-encoded problem – typically a local search engine like WalkSAT (Kautz & Selman 1996) or a Davis-Putnam implementation like TABLEAU (Ernst, Millstein, & Weld 1997) – performs a search on the whole spectrum of encoded variables, branching on both action and fluent variables, without distinction. As a consequence, it searches in a space of truth assignments of size $2^{(\mathcal{A}+\mathcal{F})\cdot n}$, while Fact 2. suggests a size bound of $2^{\mathcal{A}\cdot n}$.

```
DP(Γ, U)
    if ∅ ∈ Γ then return;
    if Γ = ∅ then exit with a model of U;
    UNIT-PROPAGATE(Γ, U);
    L := a literal such that L or ¬L occurs in Γ;
    DP(Γ ∪ {L}, U);
    DP(Γ ∪ {¬L}, U);

UNIT-PROPAGATE(Γ, U)
while there is a unit clause {L} in Γ
    U := U ∪ {L};
    for every clause C ∈ Γ
        if L ∈ C then Γ := Γ \ {C}
        else if ¬L ∈ C then
                    Γ := Γ \ {C} ∪ {C \ {¬L}}
    end for
end while
```

Figure 2: The Davis-Putnam procedure

Exploiting determinism on fluent values

Consider the standard Davis-Putnam procedure described in Figure 2. Intuitively, at each step DP performs first all the deterministic assignments by invoking UNIT-PROPAGATE; then it performs a non-deterministic step by choosing a literal L, and recurses sequentially with $L = true$ and $L = false$. L can be both an action or a fluent literal.

Fact 2. in the previous section suggests that in SAT-based Planning the search should be restricted to find values to action variables only, as the values for fluent variables can be eventually obtained for free. To this extent, we introduce DP*, which is obtained from DP by substituting the literal choice step with the following:

$L :=$ an <u>action</u> literal such that L or $\neg L$ occurs in Γ;

that is, we force DP* to select only action literals like A_t^i or $\neg A_t^i$ in the non-deterministic step.

DP* terminates for every encoding used, as all fluent variables are assigned deterministically by UNIT-PROPAGATE. The idea is that all encodings create clauses with at most two fluent variables, of the form F^j_{t-1},F^j_{t+1}. If no more action variables occur in Γ, then only unit clauses like (F^j_{t-1}) and $(\neg F^j_{t-1})$ and binary clauses like $(\neg F^j_{t-1} \vee F^j_{t+1})$ and $(F^j_{t-1} \vee \neg F^j_{t+1})$ are left over, which are all solved by UNIT-PROPAGATE, from (F^j_0)'s and $(\neg F^j_0)$'s onward.

For example, assume that the regular representation of actions and the classical frame axioms have been used. For each action variable A^i_t, the SAT encoding contains the following clauses:

Regular action representation:

$$\bigwedge_{L \in preconds(A^i_t)} (\neg A^i_t \vee L) \tag{1}$$

$$\bigwedge_{L \in effects(A^i_t)} (\neg A^i_t \vee L) \tag{2}$$

Classical frame axioms:

$$\bigwedge_{F^j_{t+1} \in \mathbf{F}_{t+1}/|effects(A^i_t)|} (\neg A^i_t \vee \neg F^j_{t-1} \vee F^j_{t+1}) \tag{3}$$

$$\bigwedge_{F^j_{t+1} \in \mathbf{F}_{t+1}/|effects(A^i_t)|} (\neg A^i_t \vee F^j_{t-1} \vee \neg F^j_{t+1}) \tag{4}$$

where $|effects(A^i_t)|$ denotes the set of F^j_{t+1}'s s.t. either $F^j_{t+1} \in effects(A^i_t)$ or $\neg F^j_{t+1} \in effects(A^i_t)$.

At-least-one axiom:

$$\bigvee_{A^i_t \in \mathbf{A}_t} A^i_t \tag{5}$$

First, we notice that, if all fluent variables $F^j_{t-1} \in \mathbf{F}_{t-1}$ are assigned and one action variable A^i_t is assigned true, then UNIT-PROPAGATE assigns deterministically all $F^j_{t+1} \in \mathbf{F}_{t+1}$ by resolving A^i_t with the action clauses (2) if $F^j_{t-1} \in effects(A^i_t)$, and with the frame axiom clauses (3),(4) if $F^j_{t-1} \notin effects(A^i_t)$. Then we notice that, if it is reached a situation where no more action variables occur in Γ and no unit propagations can be performed, then all fluent variables have already been assigned, and thus it is reached either a solution or a backtrack condition. In fact, the at-least-one axiom clauses (5) ensure that at least one A^i_t has been assigned true, for all t. Thus all fluent variables have been assigned, from those in $\mathbf{F}_0$ onward.

On the whole, DP* searches in a space of truth assignments of size $2^{\mathcal{A}\cdot n}$, allowing a reduction of a $2^{\mathcal{F}\cdot n}$ factor, as predicted in the previous section.

Experimental Results

In this Section we report on some comparative analysis we have performed using the MEDIC system for the SAT problem generation, and TABLEAU and TABLEAU* as propositional solvers.[3] TABLEAU is an efficient implementation of the Davis-Putnam procedure (Crawford & Auton 1996). TABLEAU* is exactly the same as TABLEAU except that is has been modified –as described in the previous section– in order to split only over action variables.[4]

More in detail, we considered a suite of planning problems as encoded and distributed in the MEDIC system. Most of these problems are the same analyzed in (Ernst, Millstein, & Weld 1997). For each problem, we generated with MEDIC the corresponding SAT-problem for each of the six encodings we have considered. Finally, we ran TABLEAU and TABLEAU* on these encodings. Table 1 summarizes the results. For each run, Table 1 reports both the CPU time (top line) and the number of nodes of the search trees of the two systems (bottom line) on the first of the satisfiable instances generated by MEDIC. (Notice that we do not take into consideration the time that MEDIC took to generate the SAT problems.) A "–" means that the system has been stopped after 2700s of CPU time. For convenience, we plot the CPU time for the erpe and ecse encodings (i.e. the two encodings using the regular action representation) in Figure 3. In this figure

- TABLEAU's and TABLEAU*'s values are represented by the dashed and the black bar respectively.
- An apparently missing line (like TABLEAU* CPU time for FIXA-2 in Figure 3, left) corresponds to a value too small to appear on the plot.

Notice the logarithmic scale on the horizontal axis of the figure.

As it can be seen, TABLEAU* (when it terminates within the time limit) performs better than TABLEAU on any problem of a certain size. Notice that the cases where TABLEAU performs better than TABLEAU* are very few, and the corresponding gaps are minor. If we limit our attention to the CPU times, we see that:

- For the erpe encodings (Figure 3, left and Table 1, second column), TABLEAU* always performs better than TABLEAU. The gap between the two can go up till 2 orders of magnitude (for BIG-BW1 and HANOI-STRIPS3).

[3] TABLEAU* and detailed instructions about how to reproduce all our experiments can be found at `ftp://ftp.mrg.dist.unige.it/` in `pub/mrg-systems`. All the experiments have been performed on a `Pentium II 266MHz, 96MBRam`.

[4] This has been accomplished by dividing the set of variables in the SAT problem into the two separate sets of fluent variables and action variables. TABLEAU* chooses the variables to split among those in this last set.

- For the crse encodings (Figure 3, right and Table 1, third column), TABLEAU* always performs better on problems that take more than one second to be solved. The gap between TABLEAU* and TABLEAU can go up to more than three orders of magnitude (TABLEAU was not able to solve both MONKEY-TEST1 and HANOI-STRIPS3 within the time limit, while TABLEAU* solved these problems in 2.93s and 1.28s respectively).
- For the ecse encodings (Table 1, fourth column), TABLEAU* always performs better on problems that take more than one second to be solved. The gap between TABLEAU* and TABLEAU can go up to more than three orders of magnitude (TABLEAU was not able to solve MONKEY-TEST2 within the time limit, while TABLEAU* solved this problem in 0.50s).
- For the ccse encodings (Table 1, fifth column), TABLEAU* always performs better on problems that take more than one second to be solved. The gap between TABLEAU* and TABLEAU can go up to more than four orders of magnitude (TABLEAU was not able to solve SMALL-BW1 within the time limit, while TABLEAU* solved this problem in 0.08s).
- For the ebse encodings (Table 1, sixth column), the two systems perform in roughly the same way.
- For the cbse encodings (Table 1, seventh column), TABLEAU* always performs better than TABLEAU. The gap between the two can go up till 4 orders of magnitude (TABLEAU was not able to solve FIX2-STRIPS within the time limit, while TABLEAU* solved this problem in 0.41s).

Considering the values reporting the number of nodes in the search trees traversed by the two systems, we see that they reflect the values of the corresponding CPU times. The biggest gap between the two systems is obtained with the crse encoding on SMALL-BW1: TABLEAU has traversed 100,689 nodes, while TABLEAU* traversed 17 nodes.

We also see that our experiments using TABLEAU* confirm the observations in (Ernst, Millstein, & Weld 1997). Namely:

- The erpe encodings result to be the smallest and the ones which can be solved fastest.
- The explanatory encodings perform better than the corresponding classical.
- The bitwise (ebse and cbse) encodings take more time to be solved than the respective regular (erpe and crse) and simple split (ecse and ccse) encodings.

This last fact might sound surprising since the bitwise encodings greatly reduce the number of action variables. The worse behavior than the other encodings might be due to the fact that the resulting SAT problems do not contain (or contain only few) binary clauses. This last fact makes TABLEAU and TABLEAU* splitting heuristics ineffective (see (Crawford & Auton 1996)).

Conclusion

In this paper we build from a very simple idea: given an initial state, the actions to be performed are the only sources of nondeterminism in STRIPS planning problems. This is very simple, but implementing this idea in current planners via SAT has important consequences, since it greatly reduces the size of the search space searched in by the SAT decider used to solve the final SAT problem. To justify the claim, we conducted a serie of experiments, showing that we can obtain an up to 4 orders of magnitude speed up.

Of course, we do not claim that these almost always positive results are indipendent from (a) the particular problems considered, or (b) the particular STRIPS formulations, or (c) the particular method used to generate the SAT problem, or (d) the particular SAT decider used. We do not expect this to be the case. Changing any of the above four elements may lead to very different results.

Finally, it is clear that this work opens an interesting line of research. Namely, how to build efficient SAT deciders tailored for STRIPS planning problems. TABLEAU* is a first attempt to build this decider. Future work will be to explore other directions.

References

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proc. of IJCAI-95*, 1636–1642.

Crawford, J., and Auton, L. 1996. Experimental results on the crossover point in satisfiability problems. *Artificial Intelligence* 81(1-2):31–58.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*.

Green, C. 1969. Application of theorem proving to problem solving. In *Proc. IJCAI*, 219–240.

Haas, A. 1987. The case for domain-specific frame axioms. In Brown, F. M., ed., *The Frame Problem in Artificial Intelligence, Proc. 1987 Workshop.*

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. ECAI-92*, 359–363.

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. AAAI-96*, 1194–1201.

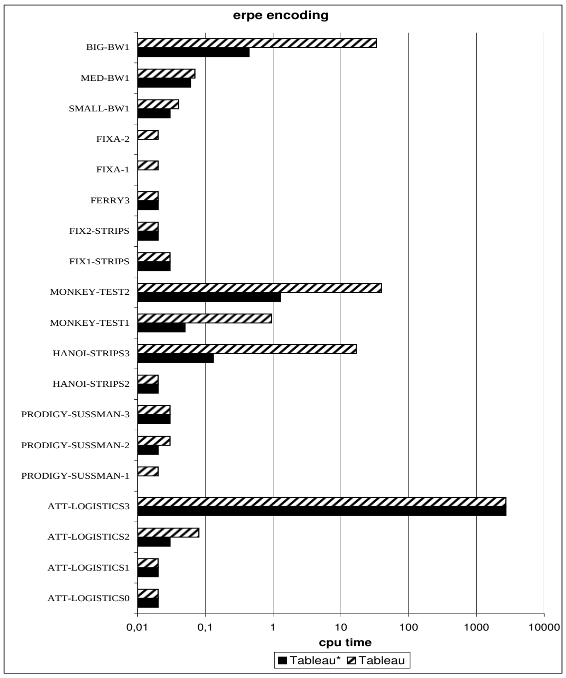
Kautz, H.; McAllester, D.; and Selman, B. 1997. Encoding plans in propositional logic. In *Proc. KR-97*, 374–384.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 4. Edinburgh: Edinburgh University Press. 463–502. Reproduced in (McCarthy 1990).

McCarthy, J. 1990. *Formalizing Common Sense: Papers by John McCarthy.* Norwood, NJ: Ablex.

DOMAIN	ERPE Tab	ERPE Tab*	CRSE Tab	CRSE Tab*	ECSE Tab	ECSE Tab*	CCSE Tab	CCSE Tab*	EBSE Tab	EBSE Tab*	CBSE Tab	CBSE Tab*
BIG-BW1	33.55	0.44	–	–	–	61.26	–	–	–	–	–	–
	12580	82	–	–	–	5738	–	–	–	–	–	–
MED-BW1	0.07	0.06	–	–	0.33	0.17	–	1.13	11	10.69	–	–
	9	1	–	–	84	22	–	112	1334	1335	–	–
SMALL-BW1	0.04	0.03	532.24	0.52	0.05	0.05	–	0.08	0.15	0.12	–	606
	3	0	100689	17	6	4	–	6	33	23	–	232431
FIXA-2	0.02	0.01	0.15	0.21	0.2	0.17	1.09	1.15	0.02	0.06	3.18	3.21
	4	4	111	146	48	39	344	347	8	59	4019	4351
FIXA-1	0.02	0.01	0.25	0.09	0.18	0.18	1.83	0.68	0.03	0.03	10.95	5.81
	4	4	191	58	43	39	648	207	8	7	21764	10746
FERRY3	0.02	0.02	0.03	0.01	0.02	0.02	0.02	0.01	0.01	0.01	0.02	0.01
	1	1	0	1	0	0	1	1	1	1	3	2
FIX2-STRIPS	0.02	0.02	0.11	0.08	0.06	0.07	0.12	0.12	0.04	0.03	2700	0.41
	11	7	2	6	2	2	5	4	3	3	–	192
FIX1-STRIPS	0.03	0.03	0.14	0.14	0.11	0.1	0.17	0.25	0.09	0.08	2700	268.47
	16	11	7	2	2	3	8	20	23	19	–	121532
MONKEY-TEST2	39.39	1.28	–	–	–	0.5	–	478.86	–	–	–	–
	46035	976	–	–	–	13	–	45646	–	–	–	–
MONKEY-TEST1	0.95	0.05	–	2.93	0.4	0.16	–	0.57	0.74	0.59	–	2.21
	1291	3	–	200	119	11	–	95	463	269	–	178
HANOI-STRIPS3	16.82	0.13	–	1.28	22.42	0.21	–	0.95	1.22	2.97	–	–
	14674	27	–	90	7047	38	–	260	364	904	–	–
HANOI-STRIPS2	0.02	0.02	0.03	0.04	0.04	0.03	0.04	0.03	0.04	0.04	43.12	0.94
	3	2	9	4	5	4	11	2	14	13	124423	2207
PRODIGY-SUSSMAN-3	0.03	0.03	0.42	0.74	0.06	0.04	5.31	0.12	1.52	0.07	–	144.01
	8	4	155	209	16	2	2863	24	928	11	–	132803
PRODIGY-SUSSMAN-2	0.03	0.02	0.06	0.05	0.03	0.03	0.18	0.04	0.05	0.04	–	1.68
	2	1	8	4	3	2	125	2	5	6	–	1992
PRODIGY-SUSSMAN-1	0.02	0.01	0.03	0.02	0.02	0.02	0.03	0.02	0.03	0.02	87.29	0.06
	0	0	0	0	0	0	0	1	1	1	285701	53
ATT-LOGISTICS3	–	–	–	–	–	–	–	–	–	–	–	–
	–	–	–	–	–	–	–	–	–	–	–	–
ATT-LOGISTICS2	0.08	0.03	2606.47	23.07	1.45	0.93	–	91.81	11.85	5.73	–	–
	91	12	944295	4900	214	115	–	20968	9922	4533	–	–
ATT-LOGISTICS1	0.02	0.02	1.69	0.84	0.21	0.21	5.68	0.66	0.23	0.04	–	326.31
	14	9	929	196	17	17	1755	176	227	9	–	404976
ATT-LOGISTICS0	0.02	0.02	0.04	0.03	0.08	0.08	0.09	0.08	0.02	0.02	259.55	0.33
	6	6	5	2	2	1	10	7	8	8	627061	773

Table 1: CPU time (top line) and nodes in the search tree (bottom line) for the six encodings



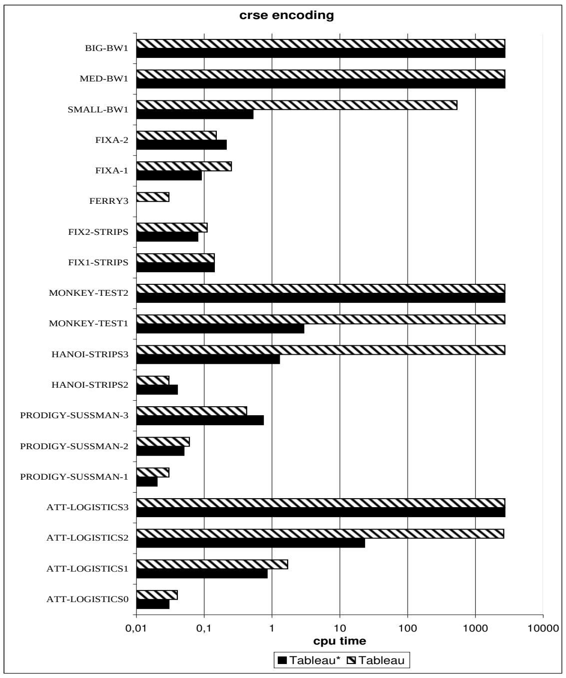


Figure 3: CPU time for erpe (left) and crse (right) encodings