Estimating Reaction Plan Size

Marcel Schoppers
Robotics Research Harvesting, PO Box 2111, Redwood City, CA 94063

Abstract

The Shannon/Ginsberg circuit size estimate, by assuming independence of Boolean inputs, is not usable as a plan size estimate. By re-estimating circuit size as a function of the number of *combinations* w of Boolean inputs, I show that a reaction plan over w world states should grow as $\mathbf{O}(w/\log w)$, on average. However, in a Blocks World containing N blocks and $w \approx N^N$ world states, actual Universal Plans grow only as $\mathbf{O}(N^3)$. This difference is shown to be attributable to the Universal Plans' use of dynamically bound object variables. Finally I obtain the general domain-independent result that for a domain containing w world states, the expected size of a reaction plan *with variables* is $\mathbf{O}((w/\log w)^{1/\log_b(p+(b-1)v)})$ where p is the number of preconditions per operator, v is the number of those preconditions that introduce an unbound variable, and b is the number of possible bindings per variable. The exponent is < 1 and allows this formula to predict plan size reductions of many orders of magnitude.

Review of Plan Size Arguments

Reaction plans are a relatively new representation for controlling embedded agents. As argued in (Schoppers 1989b), they mitigate the cost of run-time replanning by functioning as caches of responses to possible situations, thus trading time against space. Predictably, the increased space requirements have led to criticisms that for domains of any "reasonable size", a reaction plan could only prescribe reactions to arbitrary situations by being a "very large" plan indeed. This argument was made most forcibly by (Ginsberg 1989), who formalized domain size as the number of atomic propositions (or ground literals) in the vocabulary of the domain's representation, formalized plan size as the number of gates in a hardware implementation of a reaction plan, and employed a version of (Shannon 1949)'s circuit size estimate to argue that the vast majority of reaction plans for domains having n ground literals would require $\mathbf{O}(2^n/4n)$ gates for their hardware implementation, thus implying that reaction plans would in general be too large to be practical — unless they contained a reaction that invoked a planner, in which case a reasonable plan size would be obtainable only if the reaction plan "passed the buck" to planning nearly all the time (so why have the reaction plan at all).

Ginsberg's arguments were accompanied by two rebuttals. (Chapman 1989) exhibited a reaction plan of fixed size that was capable of building block towers of arbitrary height (without resorting to planning). (Schoppers 1989b) argued from common sense: Ginsberg's arguments imply also that production systems[1] are "impractical in general" because needing an exponential number of rules. Yet the AI field deems the hand-building of production systems a worth-while effort; how then can the automatic building of similar functionality be deemed impractical? Although those rebuttals should have cast some doubt on Ginsberg's argument, many subsequent papers have cited him as proving conclusively that reaction plans are "impractical in general" (Doyle & Wellman 1990, p.34) (Ingrand & Georgeff 1990, p.284) (Kaelbling 1990a, p.437) (Drummond & Bresina 1990) (Christensen 1990, p.1006) (Godefroid & Kabanza 1991, p.640) (Chrisman & Simmons 1991, p.761) (Bonasso 1991, p.1225) (Lyons & Hendriks 1992, p.154).

Before proceeding with the main argument, we must clarify the meaning of "all possible situations" as the range of situations a reaction plan should have reasonable responses for. It might mean "all physically possible situations" but then, if one cannot expect SIPE (for example) to build a plan that pushes a meddlesome baby out of the way when SIPE's domain model does not distinguish meddlesome babies, neither can one fault reaction plans for having the same limitation. If one replies that domain models must be dynamically refinable and that SIPE, given a refined model that includes meddlesome babies, could simply build new plans that deal with such babies, then similarly, both (Kaelbling 1988) and (Schoppers 1989a) describe how to automatically (re)construct reaction plans from do-

[1] In a production system the choice of rule to be fired amounts to a reaction to the contents of the system's working memory. If the rule firing engine carries any "state" from one firing to the next, that state exists solely for efficiency purposes, and makes no difference to the rule/reaction selected.

main models. From this point, the only fair arguments about reaction plan size are those that hold even under fixed domain models. Said differently, fair size arguments will hold even when "all possible situations" is reduced to "all situations distinguishable within a given domain model."

This paper

- isolates a false assumption that invalidates Shannon's circuit size estimate as a predictor of reaction plan size;
- develops an improved predictor of reaction plan size;
- exhibits a small reaction plan that defies even the improved predictor;
- mathematically attributes the unexpectedly small plan size to the plan's use of dynamically bound object variables;
- finds an expression for the expected size of reaction plans containing dynamically bound object variables;
- tests the expression on reaction plans for block stacking, and gets results that are many orders of magnitude better than previous estimates.

Importance of the Independence Assumption

The circuit size argument proceeds as follows. 1) There are 2^{2^n} Boolean functions of n Boolean variables; 2) With $g = 2^n/4n$ gates it is possible to build at most $2^{2^n/2}/n^{2g}$ ($\ll 2^{2^n}$) Boolean functions (Shannon 1949; Ginsberg 1989); hence 3) The vast majority of Boolean functions on n inputs require more than $2^n/4n$ to gates to implement them. This argument presupposes (and Ginsberg explicitly requires) that the n Boolean inputs are independent. If they are not independent, then there are $< 2^n$ possible combinations of input values, hence there are $\ll 2^{2^n}$ Boolean functions on those n inputs, hence $2^n/4n$ gates may be more than enough, hence the argument collapses. It follows immediately that the circuit size argument cannot be used to predict reaction plan size unless the independence assumption holds of the ground literals being tested by the plan.

There are not many planning domains in which the independence assumption is satisfied. In particular, fluents give rise to as many ground literals as the fluent has possible values, but those ground literals are not independent. To see how quickly fluents invalidate size estimates, observe that in an N-Blocks World, each block can sit on N things, thus generating at least N^2 ground literals, and leading to the naive conclusions that the number of world states is 2^{N^2} and the number of gates needed is $> 2^{N^2}/4N^2$. With $N = 10$ this predicts 10^{30} world states and $> 3\times10^{27}$ gates when in fact there are only 6×10^7 world states (see Derivation 1 of the Appendix). This huge discrepancy arises solely from the fact that what's-on(Block) and what's-under(Block) are fluents.

Hence, we can now conclude that the Shannon/Ginsberg argument works only in domains that contain no fluents. Common examples of fluents include the amount of fuel remaining, the distance travelled, the positions of things, and the current time. Since the world is full of fluents, the circuit size estimate is generally worthless as an estimate of plan size.

We can however salvage something of the circuit size estimate as follows:

- There are 2^w Boolean functions on w *combinations* of Boolean values.
- With g binary gates it is possible to build at most $(16(g+n+2)^2)^g$ Boolean functions on n inputs (Shannon 1949; Ginsberg 1989).
- If we happen to set $g = w/(2\log_2 w)$ and use $n \ll g$ we can build circuits for at most

$$(2^4(w/2\log_2 w)^2)^g = \frac{(2w)^{2g}}{(\log_2 w)^{2g}}$$

$$= \frac{(2^{\log_2 2w})^{2g}}{(\log_2 w)^{2g}} = \frac{2^{(\log_2 w\,+\,1)(w/\log_2 w)}}{(\log_2 w)^{(w/\log_2 w)}}$$

$$= \frac{2^{w+(w/\log_2 w)}}{(\log_2 w)^{(w/\log_2 w)}} = 2^w \times (2/\log_2 w)^{w/\log_2 w}$$

Boolean functions, which provides for less than half of the 2^w possible functions whenever $w \geq 7$.

Consequently, a planning domain containing $w \geq 7$ world states will (on average) require a reaction plan of size $\geq w/(2\log_2 w)$.

The Appendix (Derivation 2) shows that in the N-Blocks World w is $\mathbf{O}(N^N)$, so our improved plan size estimate is $\mathbf{O}(N^{N-1}/\log N)$. This estimate is shown in Table 1 under the heading "improved number of gates". Also shown, under the heading "size of random partition," is a size estimate based on the expected number of equivalence classes in a randomly selected partition on w world states (see Derivation 3). This estimate too is $\mathbf{O}(w/\log w)$ (Haigh 1972). These estimates of plan size are already much less devastating than Ginsberg's, but might still be interpreted as damaging to the (fully-explicit) reaction plans enterprise. However, the next section undermines the circuits analogy completely.

Importance of Object Variables

I begin by calculating the actual size of a Universal Plan needed to build a tower of N blocks from any initial configuration of N blocks.

Two action descriptions are required for a Blocks World that does not model the robot arm:

```
puton(X,Y) --
   on(X,Y) <+ clear(X), clear(Y).

putoff(Y,X) --
   clear(X), ontable(Y)
          <+ on(Y,X) ? clear(Y).
```

The action name comes first, followed by "--", then the action's postconditions, then "<+", and finally the action's preconditions. The preconditions preceding a "?" are filters or qualifiers on the applicability of the

blocks	number of world states	Ginsberg number of gates	improved number of gates	size of random partition	actual UP size w/o vars	actual UP size w vars	estim. UP size w vars
3	13	14	3.7	6.4	10	9	6.1
4	73	1024	12.2	23	35	22	9.6
5	501	3.36×10^{5}	58.2	107	192	45	16.0
6	4051	4.77×10^{8}	352	629	1393	81	26.0
7	37633	2.87×10^{12}	2576	4480	11834	133	40.8
8	394353	7.21×10^{16}	22078	37450	112635	204	62.5
9	4596553	7.46×10^{21}	216136	359000	1181120	297	94.0
10	5.89×10^{7}	3.17×10^{27}	0.22×10^{7}	0.39×10^{7}	1.35×10^{7}	415	140
15	6.56×10^{13}	5.99×10^{64}	1.48×10^{12}	2.30×10^{12}	8.03×10^{12}	1485	958
20	3.28×10^{20}	1.61×10^{117}	0.50×10^{19}	0.75×10^{19}	2.09×10^{19}	3630	6324
25	5.10×10^{27}	5.57×10^{184}	0.58×10^{26}	0.85×10^{26}	1.72×10^{26}	7225	41581

Table I: Sizes of Blocks World Reaction Plans.

action: If they are false, the action does not apply, and its other preconditions should not be achieved. The preconditions that come after a "?" should be achieved if they are false and the action applies. When there is no explicit "?" there is no applicability filter.

These action descriptions, when fed into a Universal Plans planner, cause the planner to autonomously discover the goal conflict associated with the Sussman Anomaly problem, and to resolve that goal conflict by generating "confinement" rules. How that is done for the 3-blocks world is described in (Schoppers 1989a). The most general confinement rule is

`on(Y,Z) < above(X,Y) <++ ¬above(Z,X)`

(where the `above(X,Y)` relation is defined as the transitive closure of the `on(X,Y)` relation). This rule informs the Universal Plans interpreter that whenever there exist bindings of `X`, `Y` and `Z` such that `on(Y,Z)`∧`above(X,Y)` is implied by the current goals and `on(Y,Z)` is false, then we must first make `¬above(Z,X)` true, after which the two goals can be achieved in the given order.

The above three rules are sufficient to drive a Universal Plans interpreter. They may also be fed into a Universal Plans compiler, to produce a fully explicit Universal Plan for a given number of blocks, as follows. For convenience, let blocks be designated by numbers rather than letters, and consider a conjunctive goal of the form `on(1,2)` ∧ `on(2,3)` ∧...∧ `on(N-1,N)` . The confinement rule reorders the conjuncts as `on(N-1,N)` < ... < `on(2,3)` < `on(1,2)` and further requires that the achievement of each conjunct should be "confined" to circumstances in which each block yet to be placed is not under the tower being built. Thus, the compiled Universal Plan would be a decision tree that has the confinement rule built in as follows (J, X, Y and Z are block numbers):

```
subplan TWR-READY(J)
{ J≤2 ?
   t: — finish top 2 blocks —
     on(1,2) ?
     t: NO-OP
     f: ACHIEVE-ON(1,2)
   f: — tower is ready from base to J —
     on(J-1,J) ?
     t: — J-1 is in place too, go up —
       TWR-READY(J-1)
     f: — J-1 not in place, unbury 1..J-2 —
       cr1-ready(J-2,J-1,J) ?
       t: cr1-ready(J-3,J-1,J) ?
       . t: ...
       .    t: cr1-ready(1,J-1,J) ?
       .       t: — not burying 1..J-2 —
       .         ACHIEVE-ON(J-1,J)
       .       — unbury each of 1..J-2 —
       .       f: ACHIEVE-NOT-ABOVE(J,1)
       .    f: ...
       . f: ACHIEVE-NOT-ABOVE(J,J-3)
       f: ACHIEVE-NOT-ABOVE(J,J-2)
}

condition cr1-ready(X,Y,Z)
{ on(Y,Z)                  — confinement not needed —
   or not above(Z,X)  — confinement achieved —
}

subplan ACHIEVE-NOT-ABOVE(Z,X)
{ ACHIEVE-CLEAR(X)
}

subplan ACHIEVE-CLEAR(X)
{ NEW VAR Y;
  Y := the-block-on X;

  clear(Y) ?
   t: PUTOFF(Y,X)
   f: ACHIEVE-CLEAR(Y)
}
```

```
subplan ACHIEVE-ON(X,Y)
{ clear(X) ?
    t: clear(Y) ?
       t: PUTON(X,Y)
       f: ACHIEVE-CLEAR(Y)
    f: ACHIEVE-CLEAR(X)
}
```

An important point: while **above(Z,X)** is a relatively complex test whose presence might allow an unfair size reduction, it turns out to be dispensible: this plan would work even if **NOT above(Z,X)** was replaced with **clear(X)**.

Observe that the plan is encoded as a few functions which invoke each other recursively; the Summary section comments further. For now we treat the above as macros and pretend that they generate a monolithic Universal Plan. The size of this plan is calculated in the Appendix (Derivation 4), and comes out to be $(N-1)^3+N$ for N blocks. This size is shown in Table 1 under the heading "actual UP size with vars".

Let us put this size in context. The Shannon/Ginsberg circuit size estimate was $\mathbf{O}(2^{N^2}/N^2)$. My improvement thereon, allowing for dependence among literals, was $\mathbf{O}(N^{N-1}/\log N)$ (taking $w \approx N^N$ from Derivation 2). The actual plan size is $\mathbf{O}(N^3)$.

I claim that the actual Universal Plan is so much smaller than the predicted sizes because it uses variables to refer to "whatever block is on top of the block to be moved". To verify this, Derivation 5 calculates how big the plan would have been if block-variables had been disallowed, and comes out with the sizes shown in Table 1 under the heading "actual UP size without vars". Observe that replacing block-variables with block-constants makes the plan larger than both my improved circuit size estimate and the partition size estimate. Consequently it is safe to say that the remarkably small size of the Universal Plan with variables can be completely attributed to the utility of the block-variables.

Expected Size of Plans Containing Variables

In this section I obtain a general domain-independent result: the size of a reaction plan containing variables as a function of the size of an equivalent plan without variables. Throughout, the size of a plan is equated with the number of leaves of the equivalent binary decision tree (which is one greater than the number of decision nodes). The expected size of the decision tree is calculated using a recurrence relation on a number of construction operations r, where each operation adds an action operator/schema instance to the tree.

Let the "average" operator instance have $p(\geq 1)$ preconditions, of which $v(\geq 0, \leq p)$ contain exactly one unbound variable (other preconditions containing none) and let all unbound variables range over $b(>1)$ possible bindings. For simplicity we assume that the subplans being used to achieve each of a given operator instance's preconditions all have the same size, and conceptually we assemble the decision tree from its leaf nodes toward its root. Then the expected size of the decision tree is given by

$$S_v(r) = \begin{cases} 1 : r = 0 \\ p\,S_v(r-1) + 1 : r > 0 \end{cases}$$

because for each tested precondition (with or without variables) a false outcome leads to the subplan of size $S_v(r-1)$ for achieving the precondition, and a true outcome leads to testing the next precondition, until all p preconditions come out true; then the last 1 is for executing the operator instance's primitive action. Thus,

$$S_v(r) = \begin{cases} r : p = 1 \\ (p^{r+1}-1)/(p-1) : p > 1 \end{cases}$$

Now suppose that the v variables in the average operator's preconditions can not be bound dynamically. Then the decision tree must explicitly test each possible binding to see if it matches the actual situation. Figure 1 shows an example from the Blocks World: the **on(X,a)** test, applied when block towers are dismantled, expands into one case in which **a** is already clear (no binding for **X**) and multiple cases in which **a** is under one of a number of possible blocks, with each block now requiring a separate subplan for its removal.

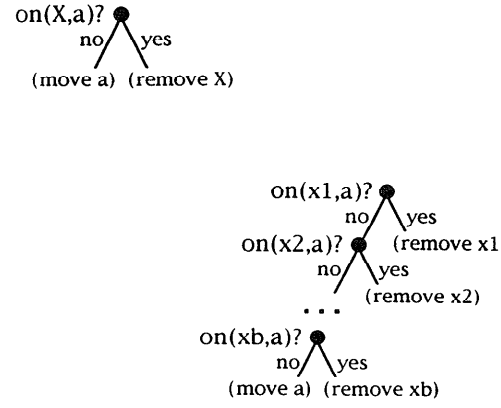

Figure 1: Plan Expansion to Eliminate Variables.

Accordingly, the size of an equivalent decision tree without variables is

$$S(r) = \begin{cases} 1 : r = 0 \\ \{v\,b + (p-v)\}\,S(r-1) + 1 : r > 0 \end{cases}$$

(wherein S has no subscript). Writing the term in braces as κ and simplifying:

$$S(r) = \begin{cases} r : \kappa = 1 \\ (\kappa^{r+1}-1)/(\kappa-1) : \kappa > 1 \end{cases}$$

Since $\kappa = 1$ requires both $1/b \leq p \leq 1$ and $v = (1-p)/(b-1)$, we ignore this case in what follows.

If the reaction plan *without* variables should distinguish S classes of world states, the appropriate number of tree construction operations is

$$r = \log_\kappa\{(\kappa-1)S+1\} - 1 \quad (\kappa > 1).$$

Using this same number of construction operations to build the equivalent decision tree *with* variables:

$$S_v = \begin{cases} \log_\kappa\{(\kappa-1)S+1\} - 1 : p = 1, \kappa > 1 \\ (p^{\log_\kappa\{(\kappa-1)S+1\}} - 1)/(p-1) : p > 1, \kappa > 1 \end{cases}$$

The case $p=1$ (one precondition per operator) reveals clearly that variables can reduce a reaction plan's size to the logarithm of of its variable-free magnitude. For the case $p>1$ we use $\log_\kappa x = \log_p x/\log_p \kappa$ to simplify:

$$S_v = \frac{\{(\kappa-1)S+1\}^{1/\log_p \kappa} - 1}{p-1} : p > 1, \kappa > 1$$

Let us check this last formula numerically. For S we use the known sizes of Universal Plans without variables. (In other domains, estimate w, then estimate S from $S^S \approx e^{w-1}$, see Derivation 3.) Assuming that the Universal Plan consists of equal numbers of the **puton** and **putoff** operators shown in Section 3, we find $p = 1.5$ and $v = 0.5$, and we set $b = N-1$, so $\kappa = (N+1)/2$. The resulting plan size estimates are shown in Table 1 under "est. UP size with vars". The results reveal that our general formula is inaccurate, under-estimating for small N and over-estimating for large N. Indeed, a little algebra (with $w \approx N^{0.8N}$, $S \approx 2w/\log_2 w$, and $1/\log_p \kappa \rightarrow \log_N p$) shows that S_v is $\mathbf{O}(p^{0.8N})$ and hence remains exponential, where the actual plan grows as N^3. Nevertheless, our formula is clearly a very large improvement on previous plan size estimates.

It is worth noting that, using the values of p and v just given, the the exponent simplifies to

$$0.585/\log_2\{(N+1)/2\}.$$

When $N=25$ this is ≈ 0.158 and is responsible for reducing the estimated plan size from 10^{26} to 10^4. While the exact reduction clearly depends on the values of p, v and b being used, the example demonstrates the power of a fractional exponent and promises similar effectiveness in other domains, since the general form of the exponent is domain-independent.

Summary

This paper has demonstrated that Shannon's circuit size estimate cannot be used as an estimate of reaction plan size, for two reasons. First, Shannon's circuit size estimate relies on an assumption that the inputs to the circuit are independent. The penalty for failing to observe the independence assumption is a massive over-estimate; the reward for observing it is the realization that virtually all planning domains contain fluents, which falsify the assumption. Second, most approaches to building reaction plans allow the plans to contain variables that are bound at execution time — a feature for which the circuit size estimate has no analog.

To drive the point home in a constructive way, I showed that the small size of a Universal Plan for Blocks Worlds was completely accounted for by the presence of dynamically bound variables in the plan, and found that in any domain, the expected size of reaction plans containing variables was $\mathbf{O}((w/\log w)^{1/\log_p(p+(b-1)v)})$, where w is the number of world states and the exponent encodes the power of dynamically bound variables. The new formula was empirically tested on Blocks World plan sizes and improved on previous estimates by many, many orders of magnitude. More generally, the new formula's fractional exponent was shown to be very effective indeed at keeping plan size under control. (Whether the new formula accurately predicts actual plan sizes is less important than the fact that the formula is based on domain-independent considerations and shows variables to be very effective.)

The fully explicit block-stacking circuits produced by other approaches will be even smaller than the sizes estimated for fully explicit Universal Plan. Those savings are generally obtained by one of two means: (1) the hand-coded "plans" may be iterative because they are built by hand, whereas Universal Plans are recursive because they are built automatically; (2) the hand-coded plans may use internal data ("state bits") to control reaction plan execution, whereas the Universal Plan analyzed above was assumed to be completely sensor-driven.

Finally, bear in mind that reaction plans are usually not fully explicit. The Universal Plans interpreter, given just the three rules required for block stacking (see Section 3), can back-chain recursively (without searching) to dynamically construct exactly the same decision sequence as the plan analyzed above. The PRS (Georgeff et al 1987) and RAPS (Firby 1987) interpreters similarly back-chain recursively to dynamically instantiate arbitrarily large plans implicit in a small number of "knowledge sources". The GAPPS compiler (Kaelbling & Rosenschein 1990b, sec 3.3.2), when made part of a situated automaton, could dynamically instantiate only the relevant parts of GAPPS programs. When fast execution-time re-assembly and disassembly of relevant plan parts is supported, the size of fully explicit reaction plans becomes entirely irrelevant.

Beyond combating an ugly rumor about reaction plans, this paper also contributes to analyses of the space complexity of solutions to entire problem domains. Such solutions include production systems and many ordinary hand-written programs — the latter may be construed as reacting to input data and the states of CPU and memory.

Appendix: Derivations

Derivation 1: Number of World States

The number of ways N blocks can be stacked is calculated in three steps:

1. find all possible combinations of tower sizes, under the assumption that blocks are identical and interchangeable,
2. for each allowed combination of tower sizes, make blocks non-interchangeable and find all ways of permuting them such that the resulting world states are distinct,
3. find the sum, over all possible combinations of tower sizes, of the number of distinct block permutations.

Step #1 is an instance of the famous "integer partition" problem (Knuth 1968, p.12). To clarify the nature of the partitions, here are the partitions for the first few integers.

N	list of partitions
1	1
2	2, 1+1
3	3, 2+1, 1+1+1
4	4, 3+1, 2+2, 2+1+1, 1+1+1+1
5	5, 4+1, 3+2, 3+1+1, 2+2+1, 2+1+1+1, 1+1+1+1+1

To generate the partitions of N+1 from those of N, add 1 to each component of each partition of N, store the resulting partition if it has not been encountered before, and finally add a partition consisting of N+1 1s.

For Step #2 we represent each partition as consisting of n_i is (e.g. a partition of 5 as 1 3s and 2 1s), then count $N!/\prod_i n_i!$ distinct states under the given partition. The denominator factors out states that differ only by a positional interchange of towers of the same size. Step #3 sums the results of Step #2 over all the partitions of N.

The above calculation was implemented as a C program. The results for $N \leq 25$ are shown in Table 1 under the heading "actual world states".

Derivation 2: Approximate Number of States

$N!$, which appears in Derivation 1, is a lower bound on the number of N-Blocks World states. A loose upper bound may be found by noting that the N things possibly under a given block can be distinguished with $\log_2 N$ bits, so that it takes only $N \log_2 N$ bits to encode what's actually under each of the N blocks in a given state, thus allowing at most $2^{N \log_2 N} = N^N$ possible states in the N-Blocks World. Empirically we find that $(N^N)^{0.8}$ is a very good approximation.

Derivation 3: Size of Random Partition

Let $\mathcal{S}_m^{(w)}$ denote a "Stirling number of the second kind" (Knuth 1968, p.65); these numbers indicate how many ways w objects can be partitioned into m non-empty subsets. Let $\mathcal{B}(w)$ be a "Bell number"; these numbers indicate how many ways a set of w objects can be partitioned. Let $c(w)$ be the cardinality of a randomly chosen partition of w objects. Then, using a well-known identity,

$$\begin{aligned} c(w) &= \frac{\sum_{i=0}^{w} i\,\mathcal{S}_i^{(w)}}{\mathcal{B}(w)} \\ \mathcal{B}(w+1) &= \sum_{i=0}^{w+1} \mathcal{S}_i^{(w+1)} = \sum_{i=0}^{w+1} (i\,\mathcal{S}_i^{(w)} + \mathcal{S}_{i-1}^{(w)}) \\ &= (c(w)+1)\,\mathcal{B}(w). \end{aligned} \quad (1)$$

An alternative formulation of $c(w)$ is

$$c(w) = \frac{1}{\mathcal{B}(w)} \sum_{i=0}^{w-1} \binom{w-1}{w-1-i} (c(i)+1)\mathcal{B}(i).$$

Here we define $c(w)$ by adding the new w^{th} element to any subset out of a partition on $w-1$ elements. A set of $w-1$ elements contains $\binom{w-1}{w-1-i}$ possible subsets of size $w-1-i$. For each such subset, the remaining i elements can be partitioned in $\mathcal{B}(i)$ ways, and for each such partition the expected number of equivalence classes is $c(i)$+1 (the additional 1 accounts for the class containing the new w^{th} element). Now, making heavy use of Equation 1, we obtain

$$\begin{aligned} c(w) &= \sum_{i=0}^{w-1} \binom{w-1}{w-1-i} \frac{\mathcal{B}(i+1)}{\mathcal{B}(w)} \\ &= \sum_{j=0}^{w-1} \binom{w-1}{j} \frac{\mathcal{B}(w-j)}{\mathcal{B}(w)} \\ &= 1 + \sum_{j=1}^{w-1} \frac{1}{j!} \frac{w-1}{c(w-1)+1} \frac{w-2}{c(w-2)+1} \cdots \frac{w-j}{c(w-j)+1} \\ &< 1 + \sum_{j=1}^{w-1} \frac{1}{j!} \left(\frac{w-1}{c(w-1)+1} \right)^j \\ &< e^{(w-1)\,/\,c(w-1)+1)} \end{aligned}$$

Raising both sides to the power $c(w-1)+1$, using $c(w) < c(w-1)+1$, and taking logarithms:

$$\begin{aligned} c(w)^{c(w-1)+1} &< e^{w-1} \\ c(w)^{c(w)} &< e^{w-1} \\ c(w) \ln c(w) &< w-1 \end{aligned}$$

An over-estimate $\hat{c}$ of $c(w)$ can now be obtained very efficiently by iterating $\hat{c} = (w-1)/\ln \hat{c}$.

Derivation 4: Size of Actual U.P.

The size of the monolithic Universal Plan (with variables) for building a tower of N blocks is the number of decision nodes, which (in any binary tree) is one less than the number of outcomes. To get that number we need to replace the macros "ACHIEVE-<goal>" with explicit decision trees containing only the puton() and putoff() actions.

- The plan for constructing a tower of N blocks numbered from 1 (at the top) to N (at the bottom) is just an instance of TWR-READY(N), which incorporates an instance of TWR-READY(N-1), and so on recursively, down to TWR-READY2.
- For J $\geq$ 3 TWR-READY(J) incorporates J-2 instances of ACHIEVE-NOT-ABOVE(Z,X), to ensure that the tower is not being built on top of blocks 1 through J-2. Unrolling the plan recursively as in the previous item, the plan incorporates N-2 + N-3 + ... + 1 = (N-1)(N-2)/2 instances of ACHIEVE-NOT-ABOVE(Z,X). Similarly, the plan incorporates N-1 instances of ACHIEVE-ON(J-1,J).
- In general, ACHIEVE-NOT-ABOVE(Z,X) could be achieved by dismantling the tower on top of Z, then moving Z to the table. Since the plan must provide for the worst case in which Z is directly on top of X and *all* the other blocks are stacked on top of Z,

ACHIEVE-NOT-ABOVE(Z,X) must be prepared to dismantle the whole tower on top of X. Moreover, everything above X must be dismantled in any case, prior to moving X. Hence we simplify the analysis by equating ACHIEVE-NOT-ABOVE(Z,X) with ACHIEVE-CLEAR(X). Then each occurrence of ACHIEVE-NOT-ABOVE(Z,X) unfolds as a series of calls to PUTOFF(Y), one for each next higher block in the tower. Since there can be at most N-1 blocks on top of X, the (N-1)(N-2)/2 occurrences of ACHIEVE-NOT-ABOVE(Z,X) select from a total of (N-1)*(N-1)(N-2)/2 outcomes.

- By virtue of the structure of TWR-READY(J) we know that the ACHIEVE-ON(J-1,J) subplan only occurs when the tower below J has been completed. Therefore, the ACHIEVE-CLEAR(J-1) and ACHIEVE-CLEAR(J) subplans incorporated within ACHIEVE-ON(J-1,J) need only cope with the J-2 and J-1 blocks (respectively) that are not already in the partially completed tower. Consequently, to dismantle the worst-case towers above blocks J-1 and J, each ACHIEVE-ON(J-1,J) subplan must unfold as (J-2)+(J-1) calls to PUTOFF(), plus one call to PUTON(J-1,J). Summing over J from 2 to N we find a total of N×(N-1) outcomes within the ACHIEVE-ON subtrees.

Thus the total number of outcomes of the decision tree is

$$(N\text{-}1)(N\text{-}1)(N\text{-}2)/2 + N(N\text{-}1) + 1 = (N\text{-}1)^3 + N.$$

(The last 1 is for the NO-OP in TWR-READY(2).) This number is shown in Table 1 under the heading "actual UP size with vars".

Derivation 5: Size of U.P. Without Variables

Recall that the actual block-stacking plan incorporated (N-1)(N-2)/2 instances of ACHIEVE-NOT-ABOVE(Z,X), each of which may have to remove up to N-1 blocks. Inability to use variables would force each such instance to cope with (N-1)! possible permutations of blocks in each tower, for a combined size of (N-1)!×(N-1)(N-2)/2 outcomes. Similarly, the N-1 instances of ACHIEVE-ON(J-1,J) each incorporate two subplans ACHIEVE-CLEAR(J) and ACHIEVE-CLEAR(J-1), which may have to remove all permutations of J-1 or J-2 blocks, respectively. From those subplans we get a size contribution of

$$\sum_{J=2}^{N} ((J-1)! + (J-2)!) = 1 + (N-1)! + 2\sum_{k=1}^{N-2} k!.$$

Adding a further N-1 cases in which ACHIEVE-ON(J-1,J) does *not* require block-removal, and the 1 case in which no action at all is necessary, we get the total sizes shown in Table 1 under the heading "est. size of UP without vars".

References

Bonasso, R.P. 1991. Integrating reaction plans and layered competences through synchronous control. Proc 12th IJCAI: 1225–1231.

Chapman, D. 1989. Penguins can make cake. *AI Magazine* 10(4): 45–50.

Chrisman, L. & Simmons, R. 1991. Sensible planning: focusing perceptual attention. Proc AAAI Nat'l Conf on AI: 756–761.

Christensen, J. 1990. A hierarchical planner that generates its own hierarchies. Proc AAAI Nat'l Conf on AI: 1004–1009.

Doyle, J. & Wellman, M. 1990. Rational distributed reason maintenance for planning and replanning of large-scale activities (preliminary report). Proc DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control: 28–36.

Drummond, M. & Bresina, J. 1990. Planning for control. Proc 5th IEEE International Symposium on Intelligent Control: 658–662.

Firby, R.J. 1987. An investigation into reactive planning in complex domains, Proc AAAI Nat'l Conf on AI: 202–207.

Georgeff, M., Lansky, A. & Bessiere, P. 1987. A procedural logic. Proc 9th IJCAI: 516–521.

Ginsberg, M. 1989. Universal planning: an (almost) universally bad idea. *AI Magazine* 10(4): 40–44.

Godefroid, P. and Kabanza, F. 1991. An efficient reactive planner for synthesizing reactive plans. Proc AAAI Nat'l Conf on AI: 640–645.

Haigh, J. 1972. Random equivalence relations. *Journal of Combinatorial Theory (A)* 13: 287–295.

Ingrand, F. & Georgeff, M. 1990. Managing deliberation and reasoning in real-time systems. Proc DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control: 284–291.

Kaelbling, L. 1988. Goals as parallel program specifications. Proc AAAI Nat'l Conf on AI: 60–65.

Kaelbling, L. 1990a. Specifying complex behavior for computer agents. Proc DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control: 433–438.

Kaelbling, L. and Rosenschein, S. 1990b. Action and planning in embedded agents. In Maes, P. (ed) *Designing Autonomous Agents.* Elsevier.

Knuth, D. 1968. *The Art of Computer Programming I: Fundamental Algorithms.* Addison-Wesley, Reading, MA.

Lyons, D. and Hendriks, A. 1992. A practical approach to integrating reaction and deliberation. Proc 1st Conference on AI Planning Systems: 153–162.

Schoppers, M. 1989a. *Representation and Automatic Synthesis of Reaction Plans.* PhD thesis, Dept of Computer Science, University of Illinois at Urbana-Champaign.

Schoppers, M. 1989b. In defense of reaction plans as caches. *AI Magazine* 10(4): 51–60.

Shannon, C. 1949. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal* 28(1): 59–98.