Calculating Criticalities*

A. Bundy[1] F. Giunchiglia[2,3] R. Sebastiani[4] T. Walsh[2,4]

[1]Dept of AI, University of Edinburgh, Edinburgh EH1 1HN, Scotland.
[2]IRST, Povo, 38100 Trento, Italy.
[3]University of Trento, Via Inama 5, 38100 Trento, Italy.
[4]DIST, University of Genoa, Viale Causa 15A, 16146 Genova, Italy.
A.Bundy@ed.ac.uk, fausto@itc.it, rseba@mrg.dist.unige.it, toby@itc.it

Abstract

We present a novel method for building ABSTRIPS style abstraction hierarchies in planning. The aim of this method is to minimize search by limiting backtracking both between abstraction levels and within an abstraction level. Previous approaches for building ABSTRIPS style abstractions have determined the criticality of operator preconditions by reasoning about plans directly. Here, we adopt a simpler and faster approach where we use numerical simulation of the planning process. We develop a simple but powerful theory to demonstrate the theoretical advantages of our approach. We use this theory to identify some simple properties lacking in previous approaches but possessed by our method. We demonstrate the empirical advantages of our approach by a set of four benchmark experiments using the ABTWEAK system. We compare the quality of the abstraction hierarchies generated with those built by the ALPINE and HIGHPOINT algorithms.

*Authors are listed in alphabetical order. The first author is supported by EPSRC grant GR/J/80702, and the last by a HCM personal fellowship. We thank Qiang Yang for assistance with ABTWEAK and HIGHPOINT, and Alessandro Coglio for assistance with the proof of the rate of convergence of the simplified PROBABILITY model.

1 Introduction

Abstraction is a powerful heuristic for tackling combinatorial complexity. Informally, it can be described as the process of mapping a representation of a problem (often called the "*ground*", or "*concrete*", representation) onto a new representation (often called the "*abstract*" representation) which is simpler to handle [GW92]. This process can be iterated to give a hierarchy of abstract spaces. The aim of abstracting a problem is to factor the search space into a set of smaller sub-spaces, ordered hierarchically by the amount of detail within them. We can then search locally within each of these spaces. We refine the solution between levels by patching the steps which do not go through. Abstraction may not, however, always reduce runtime (for example, see [BJ95]). Whilst the various overheads (*e.g.* generating the abstract space) usually have a minor impact [GW91], "thrashing" between and within abstraction levels can result in poor performance. For instance, in order to prove a goal at one level, it may be necessary to undo goals satisfied at the upper levels. This causes backtracking between levels. Such backtracking can increase runtime exponentially [Giu96]. Determining a good abstraction is therefore vital. The goal of this paper is to propose a general methodology for building abstractions automatically which minimize search by limiting both the amount of backtracking within and between abstraction levels.

The paper is structured as follows. In Section 2 we describe ABSTRIPS style abstractions and previous work in this area. We identify how ABSTRIPS style abstractions can reduce search in Section 3. We then define the theoretical properties that a method for building such abstractions should possess in Section 4. In Section 5, we propose a simple family of methods for building ABSTRIPS style abstractions based upon two simple operators for adding together criticalities. In Section 6 we prove that this family of methods has all the theoretical properties identified in Section 4. We then propose two methods for building abstractions based upon this framework in Section 7. We illustrate how our methods compute criticalities by means of four examples in Section 8. We show the empirical advantages of our methods in Section 9 using a set of four benchmark experiments with the ABTWEAK system. In each experiment, we compare the quality of the abstraction hierarchies generated with those built by two state of the art algorithms. Finally, we end with conclusions (Section 10). Some parts of this paper appear in [BGSW96].

2 ABSTRIPS

A planning problem is defined by the goal to be achieved, a set of facts true in the initial state, and a set of operators. Operators are described by a set of preconditions (*i.e.* a set of conditions which must be true for the operator to be applicable) and a set of effects. Effects are divided into adds (*i.e.* a set of facts which become true) and deletes (*i.e.* a set of facts which become false). In addition, one effect is labeled as the primary effect of an operator. Unsupervised preconditions are those that are not the effects of any operator. See Appendix A for some examples of operators.

In an ABSTRIPS style abstractions, operator preconditions are ranked according to a criticality [Sac73]. The i-th abstract space is constructed by ignoring preconditions with rank i or less. To refine a plan at the i-th level, we need to achieve those preconditions of rank i (for a formal definition of ABSTRIPS style abstractions using Green's situation calculus [Gre69] see [GW92]). ABSTRIPS style abstractions can give an exponential speed-up in the time needed to build a plan [Kno90, GW91]. However, as mentioned in the introduction, if we have to backtrack, abstraction can greatly increase the time to find a plan. The "downward refinement property" [BY94] removes the need to backtrack between abstraction levels as every abstract plan can be refined to a concrete plan. Unfortunately, relatively few abstraction hierarchies possess this property. In practice, we try to build abstractions which limit the amount of backtracking between abstraction levels but do not preclude it altogether.

Previous approaches for building ABSTRIPS style abstractions have reasoned about plans directly. For example, in ABSTRIPS [Sac73] low criticalities were assigned to those preconditions which can be achieved with short plans assuming all higher criticality preconditions are true. More recently, ALPINE reasoned about operators to build abstraction hierarchies which satisfy the "ordered monotonicity" property [Kno94]. In [BY94], Bacchus and Yang show that backtracking between abstraction levels may be needed with such abstraction hierarchies. To reduce such backtracking, they propose the HIGHPOINT procedure. This refines the abstraction hierarchies produced by the ALPINE procedure using estimates of the probability for successful refinement. The abstractions produced by HIGHPOINT are close to having the downward refinement property (in the terminology of [BY94], they are "near-DRP") but may still cause backtracking.

We propose here a novel method for building ABSTRIPS style abstractions which is both fast and simple. Instead of reasoning about plans directly, we simulate the planning process *numerically*. The simplicity of this simulation allows us to impose two simple "monotonicity" conditions not guaranteed by previous methods. These conditions ensure that harder preconditions are achieved at higher levels of abstractions. This greatly limits the amount of backtracking. To test our method empirically, we perform the complete set of experiments presented in [Kno94] and [BY94]. On each of these benchmark problems, our method gives hierarchies which offer superior performance to those generated by both the ALPINE and HIGHPOINT algorithms. We have not yet found a problem domain on which our method offers worse performance.

3 Minimizing search

To understand how abstraction can reduce search in planning, it is helpful to visualize the search space associated with a planning problem. The search space can be seen as a directed graph where nodes correspond to states and arcs correspond to operator applications. A planning problem is then to find a path from an initial node to some target node. In what follows, we confuse nodes with states, arcs with operator applications and paths with plans. The search space is a graph as there is usually more than one path between two states.

If we delete preconditions then we construct a new abstract search space. Finding a plan in the abstract search space is typically easier than finding it in the ground space as there is no need to check for the deleted preconditions. Indeed, the graph corresponding to the abstract space usually has more paths between the initial and target nodes. There are both more arcs (new operators become applicable) and more nodes (new states become achievable that are impossible in the ground space).

To refine a plan, we must satisfy the deleted preconditions. Consider a single precondition, p deleted from an operator op (our argument will generalize to multiple preconditions). Suppose our plan applies the operator to a given state s. There are two possibilities. If p holds in s then we are done. Alternatively if p does not hold then we need to find a plan from s to a new state in which p holds. In addition, we hope that we do not clobber any other preconditions along the way. Abstraction thereby restricts search in the ground space to just the subset of paths which pass through s.

If we cannot find a plan to satisfy the deleted precondition p, then we will have to backtrack to the abstract space and find an alternative path between the initial and target nodes. If there are many alternative paths, we can spend exponentially more time backtracking than planning without abstraction in the original ground space. To limit the amount of backtracking between and within abstract spaces, we therefore want to ensure that the hardest precondition are satisfied as soon as possible. In other words, we abstract the hardest preconditions in just the most abstract space. This agrees with Sacerdoti's original proposal.

> *"... literals omitted will be those that are "details" in the sense that a simple plan can be found to achieve them once the more "critical" literals have been achieved ..."* [Sac73]

However, unlike Sacerdoti, we do not consider short plans to be "simple" plans. A classic example is found in the manufacturing domain of [SP92, PS93] (see Sections 4 and 9.4 for more details). The goal of shaping, drilling and painting a steel object has a short plan but this is difficult to find. Simple plans are those that are easy to find. As it is usually too expensive to run a planner exhaustively and compare the cost of finding different plans, we need some method for approximating the cost of finding a plan. In the rest of this paper, we outline a methodology for doing this based upon simulating the planning process numerically using "criticality functions".

4 Criticality functions

To make finding the cost of plans easier, we make two simplifying assumptions. First, we assume that we are building abstractions for a changing world. We therefore consider just the operators, ignoring the specific goal to be achieved and the facts which happen to be true in the initial state. Our methods could, however, be generalized to take into account both the goal and those facts true in the initial state. Second, to simplify the numerical simulation, we apply a "granularity" abstraction [GW92] which deletes the arguments to literals. Again, this simplification could be lifted if it proved necessary for a particular domain. It is not necessary in any of the benchmark problems tested in Section 9.

Given a set of operators, Ops, we compute the criticality of the operator precondition, p by successive approximation. At the n-th iteration, the *criticality function* $C(p, n)$ returns the *numerical criticality* of p. This converges to a limiting value as we iterate n. The intuition is that the easier it is to achieve p, the smaller the numerical criticality of p should be. We collect together the limiting numerical criticalities of the same value to give the sets S_i. We then order these sets using less than, giving $S_0 < ... < S_m$. Following [Sac73], the *criticality* of a precondition, p is the index i such that $p \in S_i$. In the i-th level of abstraction, we drop all preconditions of criticality i or less. We thereby achieve the hardest preconditions in the most abstract space.

We impose various restrictions on criticality functions. There are several obvious computational properties required like totality (every precondition must have a single criticality) and convergence (numerical criticalities must converge to some limiting value). There are also various domain dependent properties. For example, criticality functions should be *order independent*. That is, they should not depend on the order we present the operators or their preconditions. This is why we described operators and their preconditions as sets. Criticality functions also ought to treat symmetric preconditions symmetrically. If swapping the precondition p for the precondition q merely reorders the operators then p and q are said to be symmetric preconditions.

Definition 1 (Symmetry) *If p and q are symmetric preconditions then then $C(p, n) = C(q, n)$.*

Whilst this property (and indeed all the following properties) are only actually required of the final limiting numerical criticalities, insisting that the property holds at each iteration n is a small burden and makes proofs much easier. We also demand that criticality functions treat equivalent effects equivalently. Let $Pre(op)$ be the preconditions of the operator op and $Ops(p)$ be the subset of operators which have p as primary effects. We say that a set of operators, S is *equivalent* to a set of operators, T iff $|S| = |T|$ (that is, the sets are the same size) and for any $op_1 \in S$ there is some $op_2 \in T$ with $Pre(op_1) = Pre(op_2)$ and vice versa.

Definition 2 (Precondition equivalence) *If $Ops(p)$ is equivalent to $Ops(q)$ then $C(p, n) = C(q, n)$.*

To reduce backtracking, we demand that the numerical criticality of a precondition decreases with the number of operators which achieve it (operator monotonicity), and increases with the number of preconditions to operators which achieve it (precondition monotonicity).

Definition 3 (Operator monotonicity) *If $Ops(p)$ is equivalent to a subset of $Ops(q)$ then $C(p, n) \geq C(q, n)$.*

We say that a set of operators, S is *subsumed* by a set of operators, T iff $|S| = |T|$ and for any $op_1 \in S$ there is some $op_2 \in T$ with $Pre(op_1) \supseteq Pre(op_2)$. Note that if S is equivalent to T then S is subsumed by T and T is subsumed by S.

Definition 4 (Precondition monotonicity) *If $Ops(p)$ is subsumed by $Ops(q)$ then $C(p,n) \geq C(q,n)$.*

If operator monotonicity is satisfied, hard preconditions (those that are primary effects of few operators) will be proved in the higher abstraction levels. This will tend to minimize backtracking. Similarly, if precondition monotonicity is satisfied, hard preconditions (those primary effects of operators with many preconditions) will be proved in the higher abstraction levels. Again this will tend to minimize the need to backtrack. Precondition and operator monotonicity both imply precondition equivalence.

Theorem 1 *Precondition or operator monotonicity implies precondition equivalence.*

Proof: In the first case, assume precondition monotonicity holds. If $Ops(p)$ is equivalent to $Ops(q)$ then $Ops(p)$ is subsumed by $Ops(q)$. Hence, by precondition monotonicity, $C(p,n) \geq C(q,n)$. But by a symmetric argument, $C(q,n) \geq C(p,n)$. Thus $C(p,n) = C(q,n)$. And this satisfies precondition equivalence. A similar argument holds in the second case for operator monotonicity. $\Box$

ALPINE and HIGHPOINT generate abstraction hierarchies which fail to satisfy these properties and therefore cause unnecessary backtracking. Consider, for example, the manufacturing domain of [SP92, PS93] listed in Appendix A. There are three operators which shape, drill and paint objects. The first operator has a single precondition `Object` and has `Shaped` as its primary effect. The second operator also has the single precondition `Object` and has `Drilled` as its primary effect. The third operator paints a steel object. It has `Object` and `Steel` as preconditions and has `Painted` as its primary effect. Precondition monotonicity ensure that the numerical criticality of `Painted` is greater or equal to that of both `Shaped` and `Drilled`. This agrees with our intuitions, as `Painted` requires an extra precondition. ALPINE, by comparison, assigns `Painted` the lowest criticality. As we will see in Section 9, this can result in a large amount of backtracking.

Note that the trivial criticality function which assigns every precondition the same numerical criticality satisfies every one of these properties. This corresponds to no abstraction levels. We therefore maximize the number of abstraction levels by treating the "greater than or equal to" relations derived from the monotonicity properties as "strictly greater than" relations wherever possible. There are many non-trivial functions which satisfy these properties. However, these properties are often sufficient to rank numerical criticalities. For example, the abstraction hierarchies generated by the methods proposed in the next Section for the examples of Section 9 follow immediately from these properties.

5 Additive criticality functions

We can identify a family of solutions by interpreting $C(p, n)$, the numerical criticality of the precondition p as the difficulty of finding a plan for p of depth 0 to n. To simplify presentation, we also introduce the numerical criticality of the operator op, $C(op, n)$ for $n > 0$. This is interpreted as the difficulty of finding a plan of depth 1 to n which ends with application of the operator op. Since the plan contains an application of op, it must be at least of depth 1.

We now define a family of additive criticality functions based upon this interpretation. In the step case, the difficulty of finding a plan for p of depth 0 to n is a function of the difficulty of finding a plan of depth 0 and of the difficulty of finding plans of depth 1 to n ending in an operator that achieves p. And the difficulty of finding a plan of depth 1 to n ending in the operator op is a function of the difficulty of finding plans of depth 0 to $n-1$ for the preconditions of op. In the base case (that is, at the zeroth iteration), we assign all preconditions the same numerical criticality, a_0.

Our definition of an additive criticality function hinges upon two "additive" operators, $\otimes$ and $\oplus$ used to add together numerical criticalities. The operator $\otimes$ determines how the criticality of a precondition is computed as a function of the criticalities of the operators that achieve it. By comparison, the operator $\oplus$ determines how the criticality of an operator is computed as a function of the criticalities of its preconditions.

Definition 5 (Additive criticality functions)

1. *$C(p, 0) = a_0$;*

2. *$C(p, n)$ is non-negative;*

3. there are two associative and commutative operators, $\otimes$ and $\oplus$ with,

$$\begin{aligned} C(p,n) &= C(p,0) \otimes C(op_1,n) \otimes ... \otimes C(op_m,n) \\ C(op,n) &= C(pre_1,n-1) \oplus ... \oplus C(pre_l,n-1) \end{aligned}$$

where $op_i \in Ops(p)$ and $pre_i \in Pre(op)$ and for $y \leq z$,

$$\begin{aligned} x \oplus y &\geq x & \qquad x \otimes y &\leq x \\ x \oplus y &\leq x \oplus z & \qquad x \otimes y &\leq x \otimes z. \end{aligned}$$

Property 1 and 2 state that every precondition is given the same initial numerical criticality, $a_0 \geq 0$. This condition can be weakened to allow different initial values provided these initial values satisfy order independence, symmetry and precondition and operator monotonicity. Property 3 is then sufficient to guarantee all the required properties like precondition and operator monotonicity continue to hold at every iteration n.

For precondition monotonicity to hold, an operator is harder if it has more preconditions. Since $\oplus$ is the operator for "adding" the numerical criticalities of preconditions to an operator, we therefore require that $x \oplus y \geq x$. And an operator is easier if it has easier preconditions. We therefore also require that $y \leq z$ implies $x \oplus y \leq x \oplus z$. For operator monotonicity to hold, a precondition p is easier if we have more operators to achieve it. Since $\otimes$ is the operator for "adding" the numerical criticalities of operators that achieve p, we therefore require that $x \otimes y \leq x$. And a precondition is easier if the operators that achieve it are easier. We therefore also require that $y \leq z$ implies $x \otimes y \leq x \otimes z$.

6 Theoretical properties

We now show that additive criticality functions satisfy the theoretical properties identified in Section 4. By simulating the planning process numerically it is easy both to identify and to prove these properties. It is more difficult to guarantee such properties in previous approaches as they reason directly with plans. To simplify proofs, we introduce some notation for repeated application of $\otimes$ and $\oplus$. If the set S contains the elements, y_1 up to y_n then,

$$\begin{aligned} x \otimes_{y \in S} y &= x \otimes y_1 \otimes ... \otimes y_n \\ \oplus_{y \in S} y &= y_1 \oplus ... \oplus y_n \end{aligned}$$

To show convergence, we first prove that, with an additive criticality function, the numerical criticality of preconditions is monotonically decreasing.

Theorem 2 $C(p, n+1) \leq C(p, n)$.

Proof: By induction on n. In the base case,

$$\begin{aligned} C(p,1) &= a_0 \otimes_{op \in Ops(p)} C(op,1) \\ &\leq a_0 \\ &= C(p,0) \end{aligned}$$

Thus $C(p, 1) \leq C(p, 0)$.

In the step case,

$$\begin{aligned} C(p,n+1) &= a_0 \otimes_{op \in Ops(p)} C(op,n+1) \\ &= a_0 \otimes_{op \in Ops(p)} (\oplus_{q \in Pre(op)} C(q,n)) \end{aligned}$$

By the induction hypothesis,

$$C(q,n) \leq C(q,n-1)$$

By repeated application of such hypotheses and the fact that $y \leq z$ implies $x \oplus y \leq x \oplus z$,

$$\oplus_{q \in Pre(op)} C(q,n) \leq \oplus_{q \in Pre(op)} C(q,n-1)$$

By repeated application of this result and the identity, $y \leq z$ implies $x \otimes y \leq x \otimes z$,

$$\begin{aligned} C(p,n+1) &= a_0 \otimes_{op \in Ops(p)} (\oplus_{q \in Pre(op)} C(q,n)) \\ &\leq a_0 \otimes_{op \in Ops(p)} (\oplus_{q \in Pre(op)} C(q,n-1)) \\ &= C(p,n) \end{aligned}$$

Thus, $C(p, n+1) \leq C(p, n)$. □

Numerical criticalities computed by an additive criticality function are therefore bounded.

Theorem 3 $C(p, n) \in [0, a_0]$.

Proof: By induction on n. In the base case, $C(p, 0) = a_0$. In the step case, the numerical criticality is monotonically decreasing. Hence $C(p, n+1) \leq C(p, n) \leq a_0$. But $C(p, n)$ is non-negative by definition. Hence $C(p, n) \in [0, a_0]$. □

Note that both ends of this bound can be achieved.

As a simple consequence of the last two theorems, the numerical criticality converges to a limiting value irrespective of the operators.

Theorem 4 *$C(p, n)$ is convergent.*

Proof: Any bounded monotonically decreasing sequence is convergent. □

Just as importantly as convergence, additive criticality functions satisfy the other properties identified in Section 4. They are order independent and symmetric since $\oplus$ and $\otimes$ are associative and commutative operators. Additive criticality function also treat equivalent preconditions equivalently.

Theorem 5 *$C(p, n)$ is precondition equivalent.*

Proof: By cases. If $n = 0$, all preconditions are assigned the same numerical criticality, a_0. Equivalent preconditions therefore have the same numerical criticality. If $n > 0$, we assume that p and q are equivalent preconditions.

$$\begin{aligned} C(p, n) &= a_0 \otimes_{op \in Ops(p)} (\oplus_{r \in Pre(op)} C(r, n-1)) \\ &= a_0 \otimes_{op \in Ops(q)} (\oplus_{r \in Pre(op)} C(r, n-1)) \\ &= C(q, n) \end{aligned}$$

□

Additive criticality functions also satisfy both the monotonicity properties. To simplify the inductive proof, we introduce a more general monotonicity property that subsumes both operator and precondition monotonicity.

Definition 6 (Monotonicity) *If $Ops(p)$ is subsumed by a subset of $Ops(q)$ then $C(p, n) \geq C(q, n)$.*

The precondition p is more difficult to achieve than the precondition q as there are fewer operators for achieving p compared to q, and the operators for achieving p each have more preconditions. Trivially, monotonicity implies both operator and precondition monotonicity.

Theorem 6 *$C(p, n)$ is monotonic.*

Proof: The proof uses induction on n. The base case is trivial as all preconditions have the same numerical criticality, a_0. In the step case, we assume that $Ops(p)$ is subsumed by a subset of $Ops(q)$. Then

$$C(p, n+1) \quad = \quad a_0 \otimes_{op \in Ops(p)} (\oplus_{r \in Pre(op)} C(r, n))$$

We compare this term for term with,

$$C(q, n+1) \quad = \quad a_0 \otimes_{op \in Ops(q)} (\oplus_{r \in Pre(op)} C(r, n))$$

As $Ops(p)$ is a subsumed by a subset of $Ops(q)$, $|Ops(p)| \leq |Ops(q)|$. Hence $C(p, n+1)$ has fewer terms in the $\otimes$ repeated sum than $C(q, n+1)$. As $Ops(p)$ is subsumed by a subset of $Ops(q)$, the preconditions of an operator achieving p are a superset of one of the operators achieving q. The common terms in the $\otimes$ repeated sum of $C(p, n+1)$ thus contain more repeated $\oplus$ terms than the corresponding terms in the repeated $\otimes$ sum of $C(q, n+1)$. Thus, by repeated application of $x \oplus y \geq x$, the common terms in the repeated $\otimes$ sum of $C(p, n+1)$ never have a smaller numerical criticality than the corresponding terms in the repeated $\otimes$ sum of $C(q, n+1)$. With fewer terms and common terms having a larger numerical criticality, by repeated application of $x \geq x \otimes y$ and $y \geq z$ implying $x \otimes y \geq x \otimes z$, the repeated $\otimes$ sum with fewer and larger terms never has a smaller numerical criticality. Hence, $C(p, n+1) \geq C(q, n+1)$. $\square$

We could define even more general properties which, instead of comparing the preconditions to operators, merely compared the numerical criticalities of the preconditions. For example, we say that a set of operators, S is *weakly equivalent* to a set of operators, T iff for any $op_1 \in S$ there is some $op_2 \in T$ with the numerical criticalities of $Pre(op_1)$ equal to the numerical criticalities of $Pre(op_2)$ and vice versa. Equivalence implies weak equivalence but not vice versa. Similar definitions could be made for weak subsumption, and weak precondition and operator monotonicity. All the theorems proved in this section would still hold under such more general definitions as the proofs depend just on the *value* of the numerical criticality of a precondition. Substituting a precondition for a different one of the same numerical criticality will therefore leave the result unaffected. However, as we demonstrate in the next sections, we do not need such a generalization to build good abstractions hierarchies for our benchmark experiments.

7 Two solutions

To calculate criticalities, we now merely need to decide on a pair of associative and commutative operators $\otimes$ and $\oplus$ for "adding" criticalities that satisfy the simple properties of an additive criticality function. Since $\otimes$ and $\oplus$ are commutative, $x \otimes y \leq x$ means that $x \otimes y \leq \min(x, y)$ and $x \oplus y \geq x$ means that $x \oplus y \geq \max(x, y)$. One of the simplest solutions treats these inequalities as equalities. That is, we define,

$$\begin{aligned} x \otimes y &= \min(x, y) \\ x \oplus y &= \max(x, y) \end{aligned}$$

The numerical criticality of a precondition is therefore the same as that of the easiest operator that achieves it. And the numerical criticality of an operator is the same as that of its hardest precondition. As min and max satisfy Property 3 of the definition of an additive criticality function, this solution has all the required theoretical properties like operator and precondition monotonicity. Unfortunately, it is not a very interesting solution as $C(p, n) = a_0$ for all p and n. We can obtain non-identical limiting criticalities if we allow different initial values. However, the final limiting criticalities will always have limited diversity as they must be a subset of the initial values. To get non-trivial solutions, we need more complex operators for $\otimes$ and $\oplus$. In Sections 7.1 and 7.2 we propose two novel solutions. The first is based on an analogy with electrical resistance whilst the second uses ideas from probability theory. We demonstrate these solutions are empirically useful in Section 9.

7.1 The Resistor model

Our first solution is based upon the notion of "resistance to change" using an analogy with electrical resistance. This solution first appeared in [BGSW96]. To capture the difficulty of achieving preconditions, we model them like resistors. The preconditions to an operator act like resistors in series. Increasing the number of preconditions makes an operator harder to apply. Treating operator preconditions like resistors in series ensures precondition monotonicity is satisfied. Operators with the same primary effects act like resistors in parallel. Increasing the number of operators with the effect p reduces the difficulty of achieving p since we have parallel paths for achieving p. Treating operators with the same effects like resistors in parallel ensures that operator monotonicity is satisfied. We shall refer to this as the RESISTOR model for computing criticalities.

As with serial resistors, the numerical criticality of an operator is thus simply the sum of the numerical criticalities of its preconditions. We therefore define,

$$x \oplus y = x + y.$$

As with electrical resistors in parallel, the numerical criticality of a precondition is thus simply the parallel sum of the numerical criticalities of the operators with this precondition as primary effects. We therefore define,

$$\frac{1}{x \otimes y} = \frac{1}{x} + \frac{1}{y}.$$

Or equivalently,

$$x \otimes y = \frac{1}{\frac{1}{x} + \frac{1}{y}}.$$

A simple induction shows that,

$$\frac{1}{x_1 \otimes \ldots \otimes x_n} = \sum_i \frac{1}{x_i}.$$

The RESISTOR model therefore satisfies the following equations.

$$\begin{aligned} C(p,0) &= a_0 && (1) \\ \frac{1}{C(p,n)} &= \frac{1}{C(p,0)} + \sum_{op \in Ops(p)} \frac{1}{C(op,n)} && (2) \\ C(op,n) &= \sum_{p \in Pre(op)} C(p,n-1) && (3) \end{aligned}$$

Note that a_0 always factors out of the final numerical criticalities. The recursive nature of these definitions naturally leads to an iterative procedure for computing numerical criticalities. The numerical criticalities defined by these equations are always rational numbers. Whilst the limiting value of a rational sequence can be irrational, in practice the limiting values are usually rational. For efficiency, we compute the numerical criticalities to some predefined accuracy and terminate computation when an iteration produces no change to the values.

We now show that this model is indeed an additive criticality function. It therefore satisfies all the theoretical properties identified in Section 4 like convergence and operator and precondition monotonicity.

Theorem 7 *The* RESISTOR *model is an additive criticality function.*

Proof: We need to verify that $\otimes$ and $\oplus$ satisfy the definition of an additive criticality function. Property 1 holds by definition. Property 2 holds as numerical criticalities correspond to resistances, and so cannot be negative. We thus merely need to check Property 3.

The operator $\otimes$ is trivially an associative and commutative operator. And $x \otimes y = x + y \geq x$. If $y \leq z$ then $x \otimes y = x + y \leq x + z = x \otimes z$.

The operator $\oplus$ is trivially a commutative operator. It is an associative operator since,

$$\frac{1}{(x \otimes y) \otimes z} = \frac{1}{x \otimes y} + \frac{1}{z} = (\frac{1}{x} + \frac{1}{y}) + \frac{1}{z} = \frac{1}{x} + (\frac{1}{y} + \frac{1}{z}) = \frac{1}{x} + \frac{1}{y \otimes z} = \frac{1}{x \otimes (y \otimes z)}.$$

As y cannot be negative, $x \otimes y \leq x$. In addition, if $y \leq z$ then $\frac{1}{y} \geq \frac{1}{z}$ and

$$x \otimes y = \frac{1}{\frac{1}{x} + \frac{1}{y}} \leq \frac{1}{\frac{1}{x} + \frac{1}{z}} = x \otimes z.$$

$\square$

Previous methods have conventionally given unsupervised preconditions, those that cannot be changed by any operator, the maximum criticality. By Equation (1), unsupervised preconditions are assigned the numerical criticality a_0 at $n = 0$. By Equation (2), their numerical criticality remains at a_0 for all subsequent n. In Section 6 we proved that a_0 is the largest numerical criticality possible for an additive criticality function. Unsupervised preconditions are therefore assigned the maximum criticality as required.

Since the RESISTOR model is an additive criticality function it converges. Indeed convergence is typically very rapid. In the domains studied in Section 8, each iteration adds approximately another decimal digit of precision. This suggests that the difference between criticalities at each iteration decreases by at least a constant factor. To explore this analytically, we developed a simple model of the Resistor model in which each operator has m preconditions (that is, for any op, $|Pre(op)| = m$) and each precondition can be achieved by l distinct operators (that is, for any p, $|Ops(p)| = l$). This gives an and-or search tree in which m is the and-branching and l is the or-branching. Under these assumptions, the numerical criticality of a precondition converges rapidly. In Appendix B, we show that the difference between successive iterations is $O((l/m)^n)$ for $l < m$, $O(1/n^2)$ for $l = m$, and $O((m/l)^n)$ for $l > m$. This supports our empirical evidence that convergence is usually very rapid, and that the difference between successive iterations tends to decrease by at least a constant factor with each iteration.

7.2 The Probability model

Our second solution is based upon a probabilistic interpretation of $C(p, n)$. We interpret $C(p, n)$, the difficulty of a precondition p as the probability that there does not exist a plan for p of depth 0 to n. As a simplifying assumption, we assume that these probabilities are statistically independent events for different p and n. Since the model is based on probabilities, we assume that the initial numerical criticality, $a_0 \leq 1$.

The preconditions to an operator behave probabilistically like conjunctive events since each must be simultaneously true. Increasing the number of events/preconditions increases the probability of a plan not existing with this operator. By comparison, operators with the same primary effects behave probabilistically like disjunctive events. Increasing the number of events/operators with the same primary effect p reduces the probability of a plan not existing that achieves p. We shall refer to this as the Probability model for computing criticalities.

As with independent and disjunctive events, the probability that no plan exists for a precondition is simply the product of the probabilities that no plan exists for any of the operators which achieve it. We therefore define,

$$x \otimes y = x \cdot y$$

As with independent and conjunctive events, the probability that no plan exists for an operator with two preconditions is simply the sum of the probabilities that no plan exists for the two preconditions less their product. We therefore define,

$$x \oplus y = x + y - x \cdot y$$

Or equivalently,

$$(1 - x \oplus y) = (1 - x) \cdot (1 - y)$$

This equation demonstrates the duality between two conjunctive events not occurring and two disjunctive events occuring. A simple induction shows that,

$$(1 - x_1 \oplus ... \oplus x_n) = \prod_i (1 - x_i)$$

The PROBABILITY model therefore satisfies the following equations.

$$\begin{aligned} C(p, 0) &= a_0 \in [0, 1] & (4) \\ C(p, n) &= C(p, 0) \cdot \prod_{op \in Ops(p)} C(op, n) & (5) \\ 1 - C(op, n) &= \prod_{p \in Pre(op)} 1 - C(p, n - 1) & (6) \end{aligned}$$

We again prove that this is an additive criticality function. It therefore satisfies all the theoretical properties identified in Section 4 like convergence and operator and precondition monotonicity.

Theorem 8 *The* PROBABILITY *model is an additive criticality function.*

Proof: We need to verify that $\otimes$ and $\oplus$ satisfy the definition of an additive criticality function. Property 1 holds by definition. Property 2 holds as numerical criticalities correspond to probabilities and so cannot be negative. We thus merely need to check Property 3.

The operator $\otimes$ is trivially an associative and commutative operator with $x \otimes y = x \cdot y \leq x$ as $0 \leq y \leq 1$. In addition, if $y \leq z$ then $x \otimes y = x \cdot y \leq x \cdot z = x \otimes z$.

The operator $\oplus$ is trivially a commutative operator. It is also associative as,

$$\begin{aligned} x \oplus (y \oplus z) &= x + (y \oplus z) - x \cdot (y \oplus z) \\ &= x + (y + z - y \cdot z) - x \cdot (y + z - y \cdot z) \\ &= (x + y - x \cdot y) + z - (x + y - x \cdot y) \cdot z \\ &= (x \oplus y) \oplus z \end{aligned}$$

In addition, $x \oplus y = x + y - x \cdot y = x + y \cdot (1 - x) \geq x$ as $(1 - x) \geq 0$. And if $y \leq z$ then $x \oplus y = x + y \cdot (1 - x) \leq x + z \cdot (1 - x) = x \oplus z$. $\square$

By Equation (4), unsupervised preconditions are assigned the numerical criticality a_0 at $n = 0$. By Equation (5), their numerical criticality remains at a_0 for all subsequent n. Unsupervised preconditions are again assigned the maximum numerical criticality, a_0 as required.

One disadvantage of the PROBABILITY model over the RESISTOR model is that the PROBABILITY model is more computationally expensive to compute. In addition, by repeatedly taking differences, errors may propagate more easily in the computation. Interestingly, the initial value a_0 does not factor out of the calculations. Because of this sensitivity to initial values, this model may be most useful when we allow preconditions to take different initial numerical criticalities, perhaps according to an estimate of their probability of being true in the initial state. In this paper, we compute numerical criticalities in the absence of any domain knowledge. We set $a_0 = 1/2$ to reflect our ambivalence about whether a given preconditions holds in the initial state. With this value, the PROBABILITY model gave very similar results to the RESISTOR model on the benchmark problems.

Since the PROBABILITY model is an additive criticality function it converges. Convergence is again typically very rapid. In the domains studied in Section 8, each iteration adds at least another digit of precision. To explore this analytically, we used the same simple model as before in which each operator has m preconditions (that is, for any op, $|Pre(op)| = m$) and each precondition can be achieved by l distinct operators (that is, for any p, $|Ops(p)| = l$). In Appendix B, we show that there exists $\rho < 1$ and m such that for $n \geq m$,

$$\frac{|C(p, n+1) - C(p, n)|}{|C(p, n) - C(p, n-1)|} < \rho$$

In other words, the difference between successive iterations decreases by at least a constant factor, ρ with each iteration.

8 Test examples

We will illustrate our approach by computing the numerical criticalities for four benchmark domains using the RESISTOR model. The PROBABILITY model computes the same abstraction hierarchies as the RESISTOR model on these domains, taking a similar number of iterations to converge on the final numerical criticalities. For reasons of space, we therefore only give the computations of both the RESISTOR and PROBABILITY models on the first domain. The criticalities computed on these domains are tested empirically in Section 9 using the ABTWEAK system. These experiments demonstrate that the abstraction hierarchies computed by the RESISTOR and PROBABILITY models tend to minimize the amount of backtracking between abstraction levels. The operators for these four domains are given in Appendix A.

8.1 Tower of Hanoi

The representation of this well known problem consists of a single unsupervised precondition `Is-peg`, and three predicates `On-small`, `On-medium` and `On-large`. There are three operators: one moves the large disk, another the medium size disk and the third the small disk. In Tables 1 and 2, we give the numerical criticalities computed by the RESISTOR and PROBABILITY models for the different preconditions in this domain. Every iteration gives approximately another decimal place of precision to the computation.

X	$C(X,n)/a_0$					
	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=\infty$
unsupervised	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
`On-Large`	1.0000	0.8750	0.8580	0.8561	0.8559	0.8559
`On-Medium`	1.0000	0.8333	0.8125	0.8106	0.8104	0.8104
`On-Small`	1.0000	0.7500	0.7333	0.7321	0.7321	0.7321

Table 1: Numerical criticalities for the Tower of Hanoi domain using the RESISTOR model.

We group these numerical criticalities together, and order them using the less than relation. Both models gives the same abstraction hierarchy. `On-Small` is assigned the lowest criticality of 0, `On-Medium` is given a criticality of 1, `On-Large` is assigned a criticality of 2, and `Is-peg` is given the highest criticality of 3. This is in line with our intuitions for this domain. The operator for moving the medium disk subsumes the operator for moving the large disk since it is has strictly fewer preconditions. The large disk is therefore more difficult to move than the medium disk. By precondition monotonicity the criticality of `On-Large` is greater than that

X	$C(X,n)/a_0$					
	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=\infty$
unsupervised	1.0000	1.000	1.0000	1.0000	1.0000	1.0000
`On-Large`	1.0000	0.9922	0.9894	0.9889	0.9888	0.9889
`On-Medium`	1.0000	0.9687	0.9592	0.9577	0.9575	0.9575
`On-Small`	1.0000	0.8750	0.8593	0.8574	0.8572	0.8572

Table 2: Numerical criticalities for the Tower of Hanoi domain using the PROBABILITY model.

of `On-Medium`. Similarly the medium disk is more difficult to move than the small disk. `On-Medium` is therefore given a greater criticality that `On-Small`.

8.2 Robot-Box domain

This domain comes from [BY94] and is a variant of the well-known ABSTRIPS robot domain [Sac73]. The robot can either carry or pull boxes between one of six rooms. The doors connecting rooms may be either open or closed. Closed doors may be either openable or not openable. A typical configuration is given in Figure 1. In Table 3, we give the

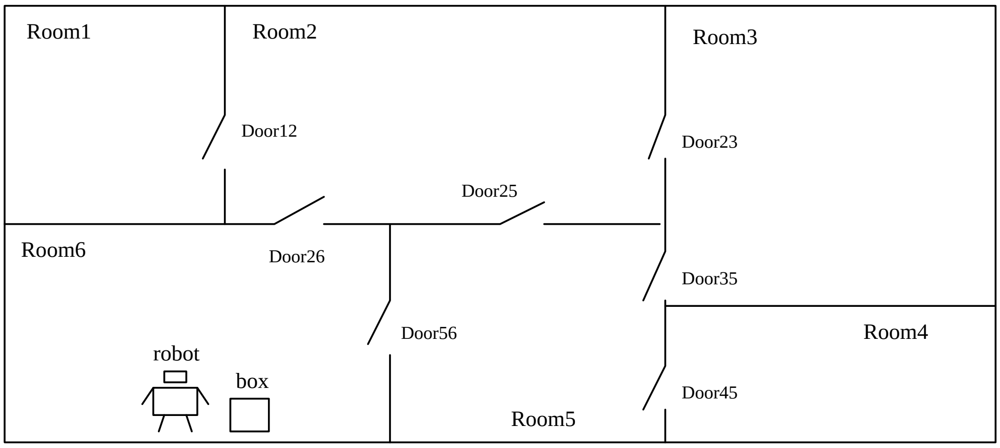


Figure 1: The robot-box domain.

numerical criticalities computed by the RESISTOR model for the different preconditions in this domain. The unsupervised preconditions are

`Connects`, `Is-Box`, `Is-Door`, `Is-Room`, and `Openable`. As in the Tower of Hanoi domain, every iteration gives approximately another decimal place of precision to the computation.

X	$C(X,n)/a_0$					
	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=\infty$
unsupervised	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
`Box-In-Room`	1.0000	0.8000	0.7830	0.7812	0.7810	0.7810
`Open`	1.0000	0.7500	0.7333	0.7321	0.7321	0.7321
`Loaded`	1.0000	0.6667	0.6250	0.6190	0.6182	0.6182
`Attached`	1.0000	0.6667	0.6250	0.6190	0.6182	0.6182

Table 3: Numerical criticalities for the robot domain using the Resistor model.

As before, we group these numerical criticalities together, and order them using the less than relation. `Attached` and `Loaded` are assigned the lowest criticality of 0, `Open` is given a criticality of 1, `Box-In-Room` is assigned a criticality of 2, and the unsupervised preconditions are given a criticality of 3. Again this is in line with our intuitions for the domain. The unsupervised preconditions cannot be changed so are the most important. Getting a box into a given room is then the next most difficult state to achieve. Opening a door is the next most difficult task to perform. Finally, attaching and loading boxes have equivalent preconditions and are equally easy to achieve.

8.3 Computer hardware

This domain has four operators which print files, turn on devices, plug devices into power outlets, and transfer files onto computers [BY94]. In Table 4, we give the numerical criticalities computed by the Resistor model for the different preconditions in this domain. The unsupervised preconditions are `CableCanReach`, `Functional`, `IsComputer`, `IsOutlet`, and `IsPrinter`. As in the previous domains, every iteration gives approximately another decimal place of precision to the computation.

We group these numerical criticalities together, and order them using the less than relation. `Loaded` is assigned the lowest criticality of 0, `PowerOn` is given a criticality of 1, `PluggedIn` is assigned a criticality of 2, `Printed` is given a criticality of 3 and the unsupervised preconditions are

X	$C(X,n)/a_0$					
	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=\infty$
unsupervised	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
`Printed`	1.0000	0.8333	0.8000	0.7949	0.7946	0.7946
`PluggedIn`	1.0000	0.6667	0.6667	0.6667	0.6667	0.6667
`PowerOn`	1.0000	0.6667	0.6250	0.6250	0.6250	0.6250
`Loaded`	1.0000	0.6667	0.6250	0.6190	0.6190	0.6190

Table 4: Numerical criticalities for the computer hardware domain using the RESISTOR model.

given the highest criticality of 4. This is again in line with our intuitions for this domain. The unsupervised preconditions cannot be changed so must be achieved in the most abstract space. The next hardest precondition to achieve is `Printed` since we must have a computer and printer turned on, and the file to print loaded on the computer. As we must plug in a device before turning it on, `PluggedIn` is assigned a greater numerical criticality than `PowerOn`. Finally, as loading a file onto a computer is less important than getting computers and printers plugged in and turned on, `Loaded` is given the lowest numerical criticality.

8.4 Manufacturing

We return to the manufacturing domain of [SP92, PS93] in which there are three operators which shape, drill and paint various objects from stock. Table 5 gives the numerical criticalities computed by the RESISTOR model for the different preconditions in this domain.

`Drilled` and `Shaped` are assigned the lowest criticality of 0, `Painted` is given a criticality of 1, and the unsupervised preconditions are given the highest criticality of 2. The `Shaped` and `Drilled` preconditions are equivalent and should be placed at the bottom of the abstraction hierarchy. The `Painted` precondition appears above them as the operator for achieving it has an additional unsupervised precondition. By precondition monotonicity, `Painted` is therefore given a greater criticality. This hierarchy agrees with the suggestions of Smith and Peot in [SP92].

X	$C(X,n)/a_0$			
	$n=0$	$n=1$	$n=2$	$n=\infty$
unsupervised	1.0000	1.0000	1.0000	1.0000
`Painted`	1.0000	0.6667	0.6667	0.6667
`Shaped`	1.0000	0.5000	0.5000	0.5000
`Drilled`	1.0000	0.5000	0.5000	0.5000

Table 5: Numerical criticalities for the manufacturing domain using the RESISTOR model.

9 Empirical results

To demonstrate the empirical advantages of the criticalities computed by the two models, we ran a set of four benchmark experiments using the ABTWEAK system [YTW96], a state-of-the-art non-linear planner combining ABSTRIPS style abstractions [Sac73] with Tweak-style partial-order planning [Cha87]. In each experiment, we compared the quality of the abstraction hierarchies generated by the RESISTOR and PROBABILITY models with those built by the ALPINE and HIGHPOINT algorithms [Kno94, BY94]. These are two of the best available procedures for generating abstraction hierarchies. Recall that the abstraction hierarchies computed by the PROBABILITY model on these four examples were identical to those computed by the RESISTOR model. The results of this section therefore also apply to the PROBABILITY model (except that the CPU time needed to compute the criticalities is, of course, slightly different).

The four experiments use standard benchmark problems taken from the literature. The first domain appears in [Kno94] and [YTW96]. The next three are presented in [BY94]. We either repeated exactly the same experiments (for example, in the manufacturing domain), or we run them in a more exhaustive manner (for example, in the robot-box domain). We used two different measurements to evaluate the performance of ABTWEAK with the different abstraction hierarchies: CPU time and the number of nodes expanded. The later is often a more reliable measurement of performance. All experiments were on a SUN Sparc 10 workstation with 32Mbytes RAM running compiled `Allegro CL 4.2` under the `Solaris 2` operating system[1].

[1]Code used in these experiments can be found at `ftp://ftp.mrg.dist.unige.it/`

9.1 Tower of Hanoi

The goal is to move a pile of three disks of different sizes from one peg to another using a third intermediate peg. At no time is a larger disk allowed to sit on a smaller one. Recall that the representation consists of an unsupervised type predicate `Is-peg`, and three predicates `On-small`, `On-medium` and `On-large`. ALPINE, HIGHPOINT and RESISTOR all produced the same abstraction hierarchy in which preconditions are abstracted according to their size. Thus, in the most abstract space, we just consider the large disk. In the next level of abstraction, we consider both the medium and large disks. And in the ground space, we consider all the disks. ALPINE generates this hierarchy in 0.01s, RESISTOR in 0.06s, and HIGHPOINT in 7.79s. Similar abstraction levels are generated for problems with more disks. In [Kno90], Knoblock shows that such abstraction hierarchies reduce a breadth first search from exponential to linear. To determine the savings possible in practice, we ran an experiment with and without abstraction. Using abstraction, the Tower of Hanoi was solved in 11.56 seconds, expanding out 57 nodes. Without abstraction, the Tower of Hanoi took more than three times as long to be solved; ABTWEAK used 38.5 seconds and expanded 379 nodes before finding a solution.

9.2 Robot-box domain

For this domain, both ALPINE or HIGHPOINT return criticalities which are order dependent. The lowest three preconditions can be permuted by reordering the operators. This is because ALPINE constructs a partial order on preconditions which is then topologically sorted. To compare results, we used the ordering of operators which generates the same abstraction hierarchy as in [BY94].

We ran experiments with both "easy" and "hard" problems. In the first set of experiments, all doors are openable. HIGHPOINT then constructs the same abstraction hierarchy as ALPINE . The criticalities are given in Table 6. ALPINE took 0.01s, HIGHPOINT 22.32s and RESISTOR 0.18s to generate these hierarchies. We ran ABTWEAK on all 30 possible goals of moving between different rooms using these criticalities. Table 7 shows that while RESISTOR performs marginally better than ALPINE and HIGHPOINT, the differences between the hierarchies are not significant as backtracking is never needed.

---

in `/pub/mrg-usr/rseba/sources/criticalities`.

ALPINE / HIGHPOINT	
4	Connects Is-Box Is-Door Is-Room Openable
3	Box-In-Room
2	Attached
1	Loaded
0	Open

RESISTOR	
3	Connects Is-Box Is-Door Is-Room Openable
2	Box-In-Room
1	Open
0	Attached Loaded

Table 6: Criticalities for the "easy" robot-box domain.

plan	mean CPU times (secs)		mean nodes expanded		samples
length	ALP/HIGH	RESIST	ALP/HIGH	RESIST	
3	0.66	0.62	28.86	27.86	14
6	4.24	4.07	143.64	142.64	14
8	12.95	12.55	379.50	378.50	2

Table 7: The "easy" robot-box domain with unlocked doors.

In the harder set of experiments, certain doors are locked. As in the RESISTOR model, HIGHPOINT increases the criticality of `Open` so that it is above `Attached` and `Loaded`. This reduces the probability of the robot meeting a locked door and thus the amount of backtracking. All other criticalities remain the same. ALPINE and RESISTOR return the same criticalities as before. We ran four sets of experiments. In each, `door25` and one of `door23`, `door26`, `door35` and `door56` are locked. In each case, there is just one unique path connecting any pair of rooms. For each set of experiments, we ran ABTWEAK on all 30 possible goals. In 8 out of the 120 problems, ABTWEAK exceeded the cut off bound of 2000 nodes using the HIGHPOINT and RESISTOR abstraction hierarchies. Using the ALPINE hierarchy, an additional problem also failed. The results are given in Table 8.

On this harder domain, the RESISTOR and HIGHPOINT hierarchies give similar results. Both perform significantly better than the ALPINE hierarchy as there is less backtracking caused by meeting locked doors. The poor mean performance of the ALPINE hierarchy was, in fact, en-

plan length	mean CPU times (secs)			mean nodes expanded			samples
	ALP	HIGH	RESIST	ALP	HIGH	RESIST	
3	0.91	0.74	0.82	28.30	28.30	27.30	40
6	6.20	5.33	5.32	162.92	160.32	159.32	40
8	39.42	29.03	28.99	775.08	752.67	751.67	24
10	82.58	69.64	67.45	1729.78	1654.87	1653.87	7/8/8

Table 8: The "hard" robot-box domain with two locked doors.

tirely due to a small number of problems where ABTWEAK backtracked extensively.

9.3 Computer hardware

In the computer hardware domain of [BY94], the goal is to print a file in an environment where there are a number of computers and printers. Computers and printers may not be turned on, may not be functional, or located near to a power outlet. As in [BY94], we ran experiments in a domain in which at the initial situation just one computer and printer are within reach of a power outlet. The criticalities generated by the different methods are given in Table 6. ALPINE took 0.01s, HIGHPOINT 15.26s and RESISTOR 0.12s to generate these hierarchies. As in [BY94], we ran ABTWEAK on 30 different problems involving between 1 and 3 files to print, and with between 1 and 10 computers, using a time limit of 1800 seconds. The results are given in Figures 2 to 4.

ALPINE		HIGHPOINT		RESISTOR	
4	`Cable-Can-Reach`	3	`Cable-Can-Reach`	4	`Cable-Can-Reach`
	`Functional`		`Functional`		`Functional`
	`Is-Computer`		`Is-Computer`		`Is-Computer`
	`Is-Printer`		`Is-Printer`		`Is-Printer`
	`Is-Outlet`		`Is-Outlet`		`Is-Outlet`
3	`Printed`	2	`Printed`	3	`Printed`
2	`Loaded`	1	`Loaded`	2	`Plugged-In`
1	`Power-On`	0	`Power-On`	1	`Power-On`
0	`Plugged-In`		`Plugged-In`	0	`Loaded`

Table 9: Criticalities for the computer hardware domain.

ALPINE performs poorly in this domain, again due to backtracking when devices are not plugged-in. RESISTOR and HIGHPOINT both require

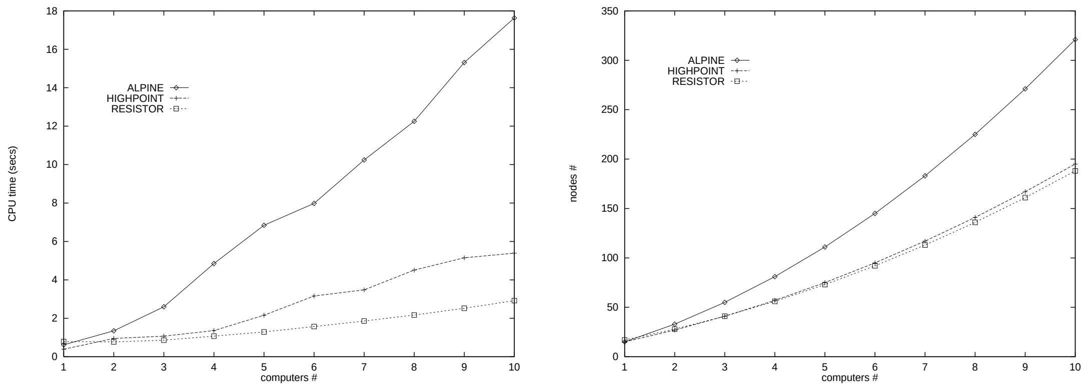


Figure 2: CPU time and nodes explored, 1 file to print.

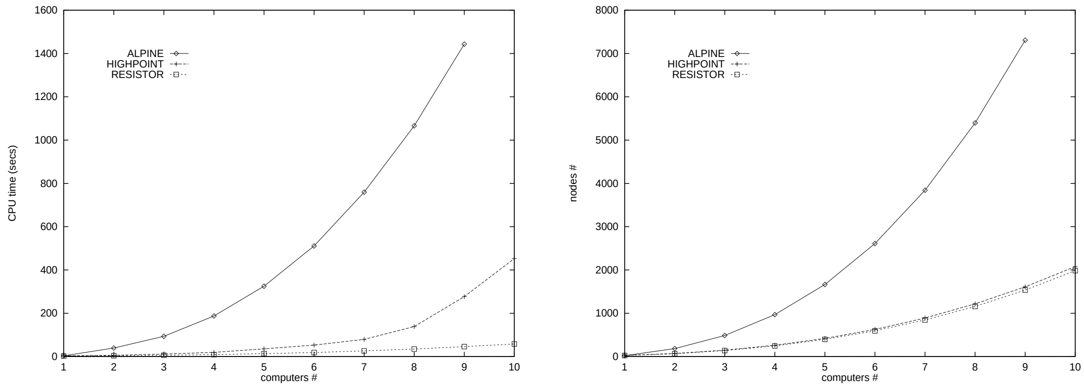


Figure 3: CPU time and nodes explored, 2 files to print.

much less backtracking. The Resistor hierarchy gives slightly better performance, most noticeably on the larger problems.

9.4 Manufacturing

We return to the manufacturing domain of [SP92, PS93]. The goal is to shape, drill and paint an object from stock. Recall that only steel objects can be painted. We assume that just one out of the large number of objects in stock are made from steel. The criticalities generated by the different methods are given in Table 10. Alpine took 0.01s, Highpoint 13.33s and Resistor 0.68s to generate these hierarchies.

Alpine 's abstraction hierarchy violates the precondition monotonicity property as the `Painted` precondition should not be lower than either the `Shaped` or `Drilled` preconditions. Highpoint compensates for the

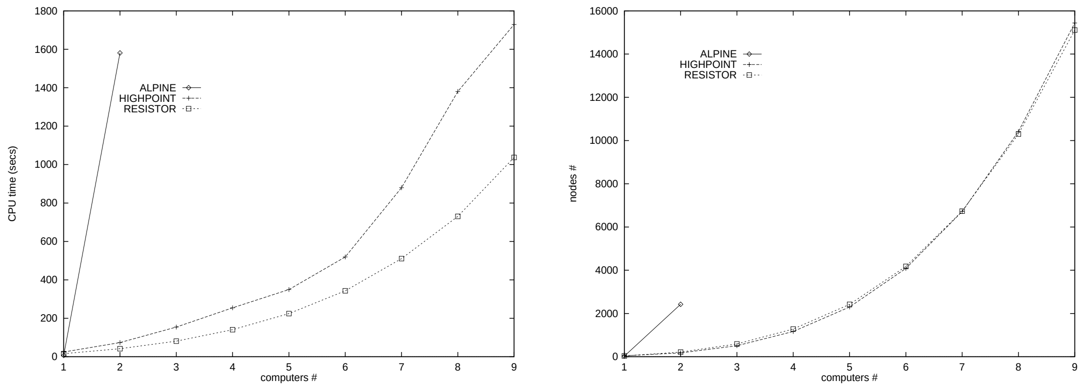


Figure 4: CPU time and nodes explored, 3 files to print.

Alpine	
3	Object Steel
2	Shaped
1	Drilled
0	Painted

Highpoint	
1	Object Steel
0	Painted Drilled Shaped

Resistor	
2	Object Steel
1	Painted
0	Shaped Drilled

Table 10: Criticalities for the manufacturing domain.

low probability of an object from stock being paintable by collapsing together the bottom three levels of ALPINE's abstraction hierarchy. This reduces the need to backtrack but gives just one level of abstraction. By comparison, RESISTOR is able to generate an additional level of abstraction.

As in [BY94], we ran ABTWEAK on problems with between 100 and 200 objects in stock. Results are plotted in Figure 5. The RESISTOR hierarchy results in less backtracking than the ALPINE hierarchy, and performs significantly better than the HIGHPOINT hierarchy due to the additional level of abstraction.

10 Conclusions

We have proposed a novel method for building ABSTRIPS style abstractions automatically based upon a simple theory of numerical criticalities. The aim of our method is to minimize the amount of backtracking within

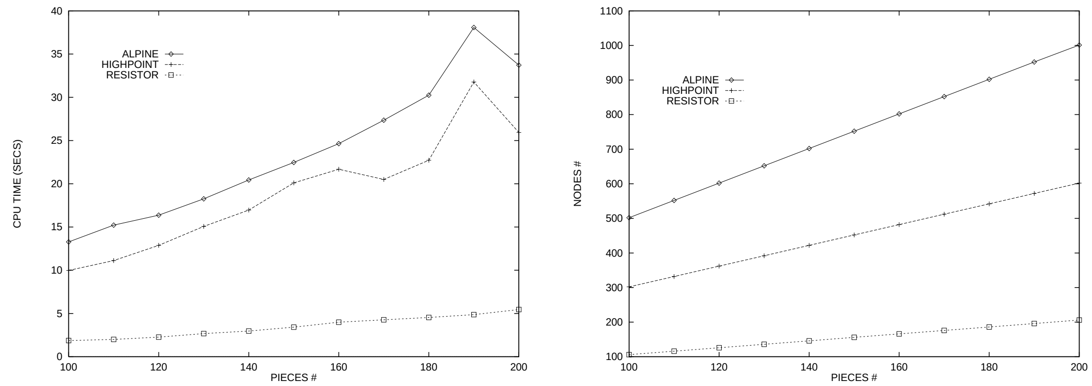


Figure 5: CPU time and nodes explored for the manufacturing domain.

and between abstraction levels. Unlike previous approaches which reasoned about plans directly, we simulate the planning process numerically. We have identified a family of solutions for building abstractions in this way based upon two general operators. The first operator computes the criticality of an operator in terms of the criticalities of its preconditions, whilst the second computes the criticality of a precondition in terms of the criticalities of the operators that achieve it. We give two examples of solutions. The first is based upon an analogy with electrical resistance, whilst the second takes ideas from probability theory. Both solutions are fast and simple to compute. The simplicity of our approach allows us to guarantee that various theoretical properties hold which are lacking in previous approaches. In particular, the abstraction hierarchies constructed by our method satisfy two simple "monotonicity" properties. These ensure that the harder preconditions are achieved in the higher abstract levels. These monotonicity properties limit the amount of backtracking required between and within abstraction levels. We have compared our method with those in the ALPINE and HIGHPOINT procedures. Using four benchmark experiments, we have demonstrated that the hierarchies constructed are better than those generated by ALPINE and HIGHPOINT. In addition, our methods build these hierarchies rapidly.

References

[BJ95] C. Backstrom and P. Jonsson. Planning with abstractions hierarchies can be exponentially less efficient. In *Proceedings of* *the 14th IJCAI*, pages 1599–1604. International Joint Conference on Artificial Intelligence, 1995.

[BY94] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1994.

[BGSW96] A. Bundy, F. Giunchiglia, R. Sebastiani and T. Walsh. Computing Abstraction Hierarchies by Numerical Simulation. To appear in *Proceedings AAAI-96*, Portland, 1996.

[Cha87] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987.

[Giu96] F. Giunchiglia. Using Abstrips abstractions – where do we stand? IRST-Technical Report, IRST, Trento, Italy, 1996.

[Gre69] C. Green. Application of theorem proving to problem solving. In *Proc. of the 1st International Joint Conference on Artificial Intelligence*, pages 219–239, 1969.

[GW91] F. Giunchiglia and T. Walsh. Using abstraction. In *Proc. of the 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, Leeds, UK, 1991. Also IRST-Technical Report 9010-08 and DAI Research Paper 515, University of Edinburgh.

[GW92] F. Giunchiglia and T. Walsh. A Theory of Abstraction. *Artificial Intelligence*, 56(2-3):323–390, 1992. Also IRST-Technical Report 9001-14, IRST, Trento, Italy.

[Kno90] C. A. Knoblock. Abstracting the Tower of Hanoi. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, pages 13–23. AAAI, 1990.

[Kno94] C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[PS93] M.A. Peot and D.A. Smith. Threat-Removal Strategies for Partial-Order Planning. In *Proceedings AAAI-93*, Washington D.C., 1993.

[Sac73] E.D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. In *Proceedings of the 3rd International Joint conference on Artificial Intelligence*, 1973.

[SP92] D. E. Smith and M. A. Peot. A Critical Look at Knoblock's Hierarchy Mechanism. In *Proc. 1st International conference Artificial Intelligence planning systems (AIPS-92)*, pages 307–308, 1992.

[YTW96] Qiang Yang, Josh Tenenberg, and Steve Woods. On the implementation and evaluation of abtweak. *Computatinal Intelligence*, Vol. 12, 1996.

Appendix A: Problem domains

The following are the operators for the problem domains used in Section 9. All the operators are taken from [YTW96, BY94]. The columns of each table give the preconditions, adds and (where appropriate) deletes respectively. The "*" symbol identifies the primary effects.

A.1 Tower of Hanoi domain

Move-large(x y)		
Is-peg(x) Is-peg(y) ¬ On-small(x) ¬ On-medium(x) ¬ On-small(y) ¬ On-medium(y) On-large(x)	On-large(y)*	¬ On-large(x)

Move-medium(x y)		
Is-peg(x) Is-peg(y) ¬ On-small(x) ¬ On-small(y) On-medium(x)	On-medium(y)*	¬ On-medium(x)

Move-small(x y)		
Is-peg(x) Is-peg(y) On-small(x)	On-small(y)*	¬ On-small(x)

A.2 Robot-Box domain

Carry-Thru-Door(b d r1 r2)		
Is-Door(d) Is-Box(b) Is-Room(r1) Is-Room(r2) Connects(d r1 r2) Loaded(b) Box-In-Room(b r1) Open(d)	Box-In-Room(b r2)*	¬ Box-In-Room(b r1)

Pull-Thru-Door(b d r1 r2)		
Is-Door(d) Is-Box(b) Is-Room(r1) Is-Room(r2) Connects(d r1 r2) Attached(b) Box-In-Room(b r1) Open(d)	Box-In-Room(b r2)*	¬ Box-In-Room(b r1)

Attach-Box(b)		
Is-Box(b) ¬ Attached(b)	Attached(b)*	

Load-Box(b)		
Is-Box(b) ¬ Loaded(b)	Loaded(b)*	

Open-Door(d)		
Is-Door(d) Openable(d) ¬ Open(d)	Open(d)*	

A.3 Computer Hardware domain

Print(file computer printer)	
Power-On(computer) Power-On(printer) Is-Computer(computer) Is-Printer(printer) Loaded(file computer)	Printed(file)*

Turn-On(device)	
Plugged-In(device) Functional(device)	Power-On(device)*

Plug-In(device outlet)	
Is-Outlet(outlet) Cable-Can-Reach(device outlet)	Plugged-In(device)*

Load(file computer)	
Is-Computer(computer) Power-On(computer)	Loaded(file computer)*

A.4 Manufacturing domain

Shape(x)		
Object(x)	Shaped(x)*	¬ Drilled(x) ¬ Painted(x)

Drill(x)		
Object(x)	Drilled(x)*	¬ Painted(x)

Paint(x)		
Object(x) Steel(x)	Painted(x)*	

Appendix B

Convergence of simple Resistor model

Recall that each operator has m preconditions and each precondition can be achieved by l distinct operators. Unfolding the definitions gives,

$$\begin{aligned} C(p,0) &= a_0 \\ \frac{1}{C(p,n+1)} &= \frac{1}{a_0} + \frac{l}{m} \cdot \frac{1}{C(p,n)} \end{aligned}$$

To identify a closed form solution, we compute the first few iterations,

$$
\begin{aligned}
\frac{1}{C(p,1)} &= \frac{1}{a_0}(1+\frac{l}{m}) \\
\frac{1}{C(p,2)} &= \frac{1}{a_0}(1+\frac{l}{m}+\left(\frac{l}{m}\right)^2) \\
\frac{1}{C(p,3)} &= \frac{1}{a_0}(1+\frac{l}{m}+\left(\frac{l}{m}\right)^2+\left(\frac{l}{m}\right)^3) \\
&\vdots
\end{aligned}
$$

A simple induction therefore shows,

$$
\frac{1}{C(p,n)} = \frac{1}{a_0}\left(\sum_{i=0}^{n}\left(\frac{l}{m}\right)^i\right)
$$

Thus,

$$
\frac{C(p,n)}{a_0} = \begin{cases} \frac{1}{n+1} & \text{if } l = m \\ \frac{1-\frac{m}{l}}{1-(\frac{m}{l})^{n+1}} & \text{if } l \neq m. \end{cases}
$$

The difference between successive iterations is therefore $O((l/m)^n)$ for $l < m$, $O(1/n^2)$ for $l = m$, and $O((m/l)^n)$ for $l > m$.

Convergence of simple Probability model

Each operator again has m preconditions and each precondition can be achieved by l distinct operators. Unfolding the definitions gives,

$$
\begin{aligned}
C(p,0) &= a_0 \\
C(p,n+1) &= a_0(1-(1-C(p,n))^m)^l
\end{aligned}
$$

To simplify notation, we write c_n for $C(p,n)$. We therefore have,

$$
c_{n+1} = c_0(1-(1-c_n)^m)^l
$$

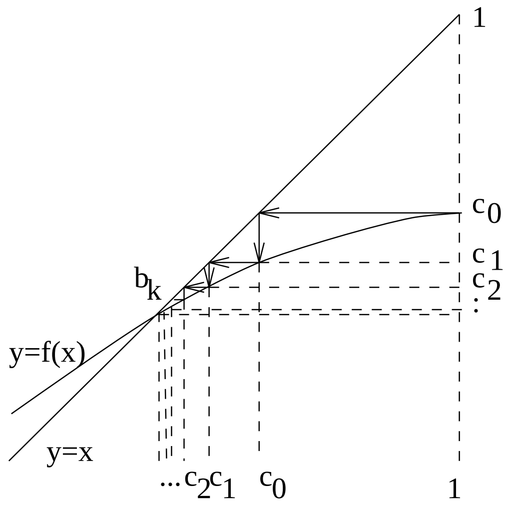
We will identify a bound on the rate of convergence of this equation by considering the fixed points of the following function,

$$
f(x) =_{\text{def}} c_0(1-(1-x)^m)^l
$$

Note that $c_{n+1} = f(c_n)$. We assume that $c_0 < 1$ since if $c_0 = 1$ then $c_n = 1$ for all n.

For all l and m, it is easy to see that $f(0) = 0$, $f(1) = c_0$ and $f(x)$ is continuous in $[0,1]$. In addition, $f'(0) = f'(1) = 0$ and $f'(x) > 0$ for all $x \in (0,1)$.

Let $b_1 < b_1 ... < b_k$ be the the fixed points of f in $[0,1]$ (i.e. $f(b_i) = b_i$). Note that $b_1 = 0$ so $k \geq 1$. The following diagram illustrates how c_n converges towards b_k, the greatest fixed point as n increases.



Since $f'(x) > 0$ in $(0,1)$, it follows that $b_k < c_0$. If $k > 1$ then for $x \in (b_{k-1}, b_k)$, $f(x) > x$. And for $x \in (b_k, 1]$, $f(x) < x$. Hence, at b_k the gradient of $y = f(x)$ must be less than that of $y = x$. In other words, $f'(b_k) < 1$. Alternatively if $k = 1$, then $f'(b_k) = 0 < 1$. By the definition of differentiation, there exists $\rho < 1$ and $b > b_k$ such that for all $x_1, x_2 \in [b_k, b]$,

$$\frac{f(x_1) - f(x_2)}{x_1 - x_2} < \rho.$$

Hence, there exists m with $c_{m-1} < b$ such that for all $n \geq m$,

$$\frac{f(c_n) - f(c_{n-1})}{c_n - c_{n-1}} < \rho.$$

That is,

$$\frac{c_{n+1} - c_n}{c_n - c_{n-1}} < \rho.$$

As c_n is monotonically decreasing,

$$\frac{|c_{n+1} - c_n|}{|c_n - c_{n-1}|} < \rho.$$

The difference between successive iterations thus decreases by at least a constant factor after the mth iteration.