The Branching Factor of Regular Search Spaces

Stefan Edelkamp
Institut für Informatik
Am Flughafen 17
79110 Freiburg
edelkamp@informatik.uni-freiburg.de

Richard E. Korf
Computer Science Department
University of California
Los Angeles, Ca. 90095
korf@cs.ucla.edu

Abstract

Many problems, such as the sliding-tile puzzles, generate search trees where different nodes have different numbers of children, in this case depending on the position of the blank. We show how to calculate the asymptotic branching factors of such problems, and how to efficiently compute the exact numbers of nodes at a given depth. This information is important for determining the complexity of various search algorithms on these problems. In addition to the sliding-tile puzzles, we also apply our technique to Rubik's Cube. While our techniques are fairly straightforward, the literature is full of incorrect branching factors for these problems, and the errors in several incorrect methods are fairly subtle.

Introduction

Many AI search algorithms, such as depth-first search (DFS), depth-first iterative-deepening (DFID), and Iterative-Deepening-A* (IDA*) (Korf 1985) search a problem-space tree. While most problem spaces are in fact graphs with cycles, detecting these cycles in general requires storing all generated nodes in memory, which is impractical for large problems. Thus, to conserve space, these algorithms search a tree expansion of the graph, rooted at the initial state. In a tree expansion of a graph, each distinct path to a given problem state gives rise to a different node in the search tree. Note that the tree expansion of a graph can be exponentially larger than the underlying graph, and in fact can be infinite even for a finite graph.

The time complexity of searching a tree depends primarily on the *branching factor* b, and the *solution depth* d. The solution depth is the length of a shortest solution path, and depends on the given problem instance. The branching factor, however, typically converges to a constant value for the entire problem space. Thus, computing the branching factor is an essential step in determining the complexity of a search algorithm on a given problem, and can be used for selecting among alternative problem spaces for the same problem.

The branching factor of a node is the number of children it has. In a tree where every node has the same branching factor, this is also the branching factor of the tree. The difficulty occurs when different nodes at the same level of the tree have different numbers of children. In that case, we can define the branching factor at a given depth of the tree as the ratio of the number of nodes at that depth to the number of nodes at the next shallower depth. In most cases, the branching factor at a given depth converges to a limit as the depth goes to infinity. This is the *asymptotic branching factor*, and the best measure of the size of the tree.

In the remainder of this paper, we present some simple examples of problem spaces, including sliding-tile puzzles and Rubik's Cube, and show how to compute their asymptotic branching factors, and how not to. We formalize the problem as the solution of a set of simultaneous equations, which can be quite large in practice. As an alternative to an analytic solution, we present an efficient numerical technique for determining the exact number of nodes at a given depth, and for estimating the asymptotic branching factor to a high degree of precision. Finally, we present some data on the branching factors of various sliding-tile puzzles.

Example Problem Spaces

The Five Puzzle

Our first example is the Five Puzzle, the 2×3 version of the well-known sliding-tile puzzles (see Figure 1). There are five numbered square tiles, and one empty position, called the "blank". Any tile horizontally or vertically adjacent to the blank can be slid into the blank position. The goal is to rearrange the tiles from some random initial configuration into a particular goal configuration, such as that shown at right in Figure 1.

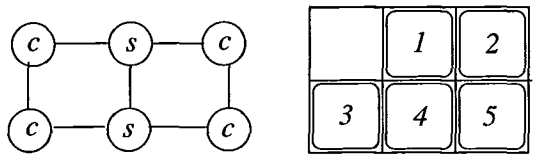

Figure 1: Side and corner states in the Five Puzzle.

The branching factor of a node in this space depends on the position of the blank. There are two different types of locations in this puzzle, "side" or s positions, and "corner" or c positions (see Figure 1). Similarly, we refer to a node or state where the blank is in an s or c position as an s or c node or state. For simplicity at first, we assume that the parent of a node is also generated as one of its children, since all operators are invertible. Thus, the branching factor of an s node is three, and the branching factor of a c node is two. Clearly, the asymptotic branching factor will be between two and three.

The exact value of the branching factor will depend on f_s, the fraction of total nodes at a given level of the tree that are s nodes, with $f_c = 1 - f_s$ being the fraction of c nodes. For a given level, f_s will depend on whether the initial state is an s node or a c node, but we are interested in the limiting value of this ratio as the depth goes to infinity, which is independent of the initial state.

Equal Likelihood The simplest hypothesis is that f_s is equal to 2/6 or 1/3, since there are two different s positions for the blank, out of a total of six possible positions. This gives an asymptotic branching factor of $3 \cdot 1/3 + 2 \cdot 2/3 = 2.333$. Unfortunately, this assumes that all possible positions of the blank are equally likely, which is incorrect. Intuitively, the s positions are more centrally located in the puzzle, and hence overrepresented in the search tree.

Random-Walk Model A better hypothesis is the following. Consider the six-node graph at left in Figure 1. In a long random walk of the blank over this graph, the fraction of time that the blank spends in any particular node will eventually converge to an equilibrium value, subject to a minor technicality. If we divide the six positions into two sets, consisting of 1, 3, and 5 verses 2, 4, and the blank in Figure 1, every move takes the blank from a position in one set to a position in the other set. Thus, at even depths the blank will be in one set, and at odd depths in the other. Since the two sets are completely symmetric, however, we can ignore this issue in this case.

The equilibrium fraction from the random walk is easy to compute. Since s nodes have degree three, and c nodes have degree two, the equilibrium fraction of time spent in an individual s state versus an individual c state must be in the ratio of three to two(Motwani & Raghavan 1995). Since there are twice as many c states as s states, c states are occupied 4/7 of the time and s states are occupied 3/7 of the time. This gives a branching factor of $3 \cdot 3/7 + 2 \cdot 4/7 \approx 2.42857$, which differs from the value of 2.3333 obtained above.

Unfortunately, this calculation is incorrect as well. While the random-walk model accurately predicts the probability of being in a particular state given a long enough random walk down the search tree, if the tree has non-uniform branching factor, this is not the same as the relative frequencies of the different states at that level of the tree. For example, consider the simple tree fragment in Figure 2. If we randomly sample the three leaf nodes at the bottom, each is equally likely to appear. However, in a random walk down this tree, the leftmost leaf node will be reached with probability 1/2, and the remaining two nodes with probability 1/4 each.

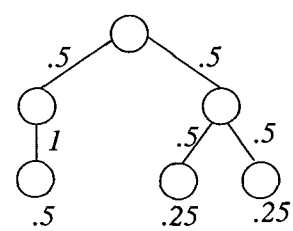

Figure 2: Tree with Nonuniform Branching Factor.

The Correct Answer The correct way to compute the equilibrium fraction f_s is as follows. A c node at one level generates an s node and another c node at the next level. Similarly, an s node at one level generates another s node and two c nodes at the next level. Thus, the number of c nodes at a given level is two times the number of s nodes plus the number of c nodes at the previous level, and the number of s nodes is the number of c nodes plus the number of s nodes at the previous level. Thus, if there are nf_s s nodes and nf_c c nodes at one level, then at the next level we will have $2nf_s + nf_c$ c nodes and $nf_s + nf_c$ s nodes at the next level. Next we assume that the fraction f_s converges to an equilibrium value, and hence must be the same at the next level, or

$$f_s = \frac{nf_s + nf_c}{nf_s + nf_c + 2nf_s + nf_c} =$$

$$\frac{f_s + 1 - f_s}{f_s + 1 - f_s + 2f_s + 1 - f_s} = \frac{1}{f_s + 2}$$

Cross multiplying results in the quadratic equation $f_s^2 + 2f_s - 1 = 0$, which has positive root $\sqrt{2} - 1 \approx .4142$. This gives an asymptotic branching factor of $3f_s + 2(1 - f_s) = 3(\sqrt{2} - 1) + 2(2 - \sqrt{2}) = \sqrt{2} + 1 \approx 2.4142$.

The assumption we made here is that the parent of a node is generated as one of its children. In practice, we wouldn't generate the parent as one of the children, reducing the branching factor by approximately one. It is important to note that the reduction is not exactly one, since pruning the tree in this way changes the equilibrium fraction of s and c states. In fact, the branching factor of the five puzzle without generating the parent as a child is 1.3532, as we will see below.

Rubik's Cube

As another example, consider Rubik's Cube, shown in Figure 3. In this problem, we define any 90, 180, or 270 degree twist of a face as a single move. Since there are six different faces, this suggests a branching factor of 18. However, it is immediately obvious that we shouldn't twist the same face twice in a row, since the same result can be obtained with a single twist. This reduces the branching factor to $5 \cdot 3 = 15$ after the first move.

The next thing to notice is that twists of opposite faces are independent of one another, and hence commutative. Thus, if two opposite faces are twisted in sequence, we restrict them to be twisted in one particular order, to eliminate the identical state resulting from twisting them in the opposite order. For each pair of opposite faces, we label one a "first" face, and the other a "second" face, depending on an arbitrary order. Thus, Left, Up and Front might be the first faces, in which case Right, Down, and Back would be the second faces. After a first face is twisted, there are three possible twists of each of the remaining five faces, for a branching factor of 15. After a second face is twisted, however, there are three possible twists of only four remaining faces, leaving out the face just twisted and its corresponding first face, for a branching factor of 12. Thus, the asympotic branching factor is between 12 and 15.

To compute it exactly, we need to determine the equilibrium frequencies of first (f) and second (s) nodes, where an f node is one where the last move made was a twist of a first face. Each f node generates six f nodes and nine s nodes as children, the difference being that you can't twist the same face again. Each s node generates six f nodes and six s nodes, since you can't twist the same face or the corresponding first face immediately thereafter. Let f_f be the equilibrium fraction of f nodes at a given level, and $f_s = 1 - f_f$ the equilibrium fraction of s nodes. Since we assume that this equilibrium fraction eventually converges to a constant, the fraction of f nodes at equilibrium must be

$$f_f = \frac{6f_f + 6f_s}{6f_f + 6f_s + 9f_f + 6f_s} =$$

$$\frac{6f_f + 6(1 - f_f)}{15f_f + 12(1 - f_f)} = \frac{6}{3f_f + 12} = \frac{2}{f_f + 4}$$

Cross multiplying gives us the quadratic equation $f_f^2 + 4f_f - 2 = 0$, which has a positive root at $f_f = \sqrt{6} - 2 \approx .44949$. This gives us an asymptotic branching factor of $15f_f + 12(1 - f_f) \approx 13.34847$.

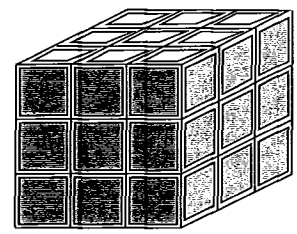

Figure 3: Rubik's Cube.

The System of Equations

The above examples required only the solution of a single quadratic equation. In general, a system of simultaneous equations is generated. As a more representative example, we use the Five Puzzle with predecessor elimination, meaning that the parent of a node is not generated as one of its children. To eliminate the inverse of the last operator applied, we have to keep track of the last two positions of the blank. Let cs denote a state or node where the current position of the blank is on the side, and the immediately previous position of the blank was in an adjacent corner. Define ss, sc and cc nodes analogously.

Figure 4 shows these different types of states, and the arrows indicate the children they generate in the search tree. For example, the double arrow from ss to sc indicates that each ss node in the search tree generates two sc nodes.

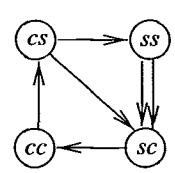

Figure 4: The graph of the Five Puzzle with predecessor elimination.

Let $n(t,d)$ be the number of nodes of type t at depth d in the search tree. Then, we can write the following recurrence relations directly from the graph in figure 4. For example, the last equation comes from the fact that there are two arrows from ss to sc, and one arrow from cs to sc.

$$\begin{aligned} n(cc,d+1) &= n(sc,d) \\ n(cs,d+1) &= n(cc,d) \\ n(ss,d+1) &= n(cs,d) \\ n(sc,d+1) &= 2n(ss,d)+n(cs,d) \end{aligned}$$

Note that we have left out the initial conditions. The first move will either generate an ss node and two sc nodes, or a cs node and a cc node, depending on whether the blank starts on the side or in a corner, respectively. The next question is how to solve these recurrences.

Numerical Solution

The simplest way is to iteratively compute the values of successive terms, until the relative frequencies of the different types of states converge. At a given search depth, let f_{cc}, f_{cs},f_{ss} and f_{sc} be the number of nodes of the given type divided by the total number of nodes at that level. Then we compute the ratio between the total nodes at two successive levels to get the branching factor. After about a hundred iterations of the equations above we get the equilibrium fractions $f_{cc} = .274854$, $f_{cs} = .203113$, $f_{ss} = .150097$, and $f_{sc} = .371936$. Since the branching factor of ss and cs states is two, and the branching factor of the others is one, this gives us the asymptotic branching factor $f_{cc} + 2f_{cs} + 2f_{ss} + 1fsc = .274854 + .406226 + .300194 + .371936 = 1.35321$. If q is the number of different types of states, four in this case, and d is the depth to which we iterate, the running time of this algorithm is $O(dq)$.

Analytical Solution

To solve for the branching factor analytically, we assume that the fractions converge to a set of equilibrium fractions that remain the same from one level to the next. This fixed point assumption gives rise to a set of equations, each being derived from the corresponding recurrence. Let b be the asymptotic branching factor. If we view, for example, f_{cc} as the normalized number of cc nodes at depth d, then the number of cc nodes at depth $d+1$ will be bf_{cc}. This allows us to directly rewrite the recurrences above as the following set of equations. The last one expresses the fact that all the normalized fractions must sum to one.

$$\begin{aligned} bf_{cc} &= f_{sc} \\ bf_{cs} &= f_{cc} \\ bf_{ss} &= f_{cs} \\ bf_{sc} &= 2f_{ss}+f_{cs} \\ 1 &= f_{cc}+f_{cs}+f_{ss}+f_{sc} \end{aligned}$$

We have five equations in five unknowns. As we try to solve these equations by repeated substitution to eliminate variables, we get larger powers of b. Eventually we can reduce this system to the single quartic equation, $b^4 - b - 2 = 0$. It is easy to check that $b \approx 1.35321$ is a solution to this equation.

While quartic equations can be solved in general, this is not true of higher degree polynomials. In general, the degree of the polynomial will be the number of different types of states. The Fifteen Puzzle, for example, has six different types of states.

General Formulation

In this section we abstract from the above examples to exhibit the general structure of the equations and their fixed point. We begin with an adjacency matrix representation P of the underlying graph $G = (V,E)$. For Figure 4, the rows P_j of P, with $j \in \{cc, cs, ss, sc\}$, are $P_{cc} = (0,1,0,0)$, $P_{cs} = (0,0,1,1)$, $P_{ss} = (0,0,0,2)$ and $P_{sc} = (1,0,0,0)$. Without loss of generality, we label the vertices V by the first $|V|$ integers, starting from zero. We represent the fractions of each type of state as a distribution vector F. In our example, $F = (f_{cc}, f_{cs}, f_{ss}, f_{sc})$. We assume that this vector converges in the limit of large depth, resulting in the equations $bF = FP$, where b is the asymptotic branching factor. In addition, we have the equation $\sum_{i \in V} f_i = 1$, since the fractions sum to one. Thus, we have a set of $|V|+1$ equations in $|V|+1$ unknown variables.

The underlying mathematical issue is an eigenvalue problem. Transforming $bF = FP$ leads to $0 = F(P - bI)$ for the identy matrix I. The solutions for b are the roots of the characteristic equation $det(P - bI) = 0$ where det is the determinant of the matrix. In the case of the Five Puzzle we have to calculate

$$det \begin{pmatrix} -b & 1 & 0 & 0 \\ 0 & -b & 1 & 1 \\ 0 & 0 & -b & 2 \\ 1 & 0 & 0 & -b \end{pmatrix} = 0$$

which simplifies to $b^4 - b - 2 = 0$.

Note that the assumption of convergence of the fraction vector and the asymptotic branching factor is not true in general, since for example the asymptotic branching factor in the Eight Puzzle of Figure

5 alternates between two values, as we will see below. Thus, here we examine the structure of the recurrences in detail. Let n_i^d be the number of nodes of type i at depth d in the tree, and n^d be the total number of nodes at depth d. Let N^d be the count vector $(n_0^d, n_1^d, ..., n_{|V|-1}^d)$. Similarly, let f_i^d be the fraction of nodes of type i out of the total nodes at depth d in the tree, and let F^d be the distribution vector $(f_0^d, f_1^d, ..., f_{|V|-1}^d)$ at level d in the tree. In other words, $f_i^d = n_i^d/n^d$, for all $i \in V$. We arbitrarily set the initial count and distribution vectors, F^0 and N^0 to one for i equal to zero, and to zero otherwise. Let the node branching factor b_k be the number of children of a node of type k, and let B be the vector of node branching factors, $(b_0, b_1, ..., b_{|V|-1})$. In terms of P the value b_k equals $\sum_{j \in V} p_{k,j}$, with the matrix element $p_{k,j}$ in row k and column j denoting the number of edges going from state k to state j. We will derive the iteration formula $F^d = F^{d-1}P/F^{d-1}B$ to determine the distribution F^d given F^{d-1}. For all $i \in V$ we have

$$\begin{aligned} f_i^d &= n_i^d/n^d \\ &= \frac{N^{d-1}(P^T)_i}{\sum_{j \in V} N^{d-1}(P^T)_j} \\ &= \frac{\sum_{j \in V} n_j^{d-1} p_{j,i}}{\sum_{j \in V} \sum_{k \in V} n_k^{d-1} p_{k,j}} \\ &= \frac{\sum_{j \in V} f_j^{d-1} n^{d-1} p_{j,i}}{\sum_{k \in V} \sum_{j \in V} f_k^{d-1} n^{d-1} p_{k,j}} \\ &= \frac{F^{d-1}(P^T)_i}{\sum_{k \in V} f_k^{d-1} \sum_{j \in V} p_{k,j}} \\ &= F^{d-1}(P^T)_i / F^{d-1}B. \end{aligned}$$

It is not difficult to prove that the branching factor of depth $d+1$ equals $F^d \cdot B$. Therefore, if the iteration formula reaches equilibrium F the branching factor b reaches equilibrium as well. In this case b equals $F \cdot B$ and we get back to the formula $bF = PF$ as cited above. Even though we have established a neat recurrence formula, up to now we have not found a full answer to the convergence of the simulation process to determine the asymptotic branching factor. A solution to this problem might be found in connections to homogenous Markov processes (Norris 1997), where we have a similar iteration formula $F^d = QF^{d-1}$, for a well defined stochastic transition matrix Q.

Experiments

Here we apply our technique to derive the branching factors for square sliding-tile puzzles up to 10×10. Table 1 plots the odd and even asymptotic branching factors in the (n^2-1)-puzzle with predecessor elimination. As n goes to infinity, the values in both columns will converge to three, the asymptotic branching factor of an infinitely large sliding-tile puzzle, with predecessor elimination.

n	n^2-1	even depth	odd depth
3	8	1.5	2
4	15	2.1304	2.1304
5	24	2.30278	2.43426
6	35	2.51964	2.51964
7	48	2.59927	2.64649
8	63	2.6959	2.6959
9	80	2.73922	2.76008
10	99	2.79026	2.79026

Table 1: The asymptotic branching factor for the (n^2-1)-Puzzle.

To understand the even-odd effect, consider the Eight Puzzle, shown in Figure 5. At every other level of the search tree, all states will be s states, and all these states will have branching factor two, once the parent of the state has been eliminated. The remaining levels of the tree will consist of a mixture of c states and m states, which have branching factors of one and three, respectively. We leave the analytic determination of the branching factor at these levels as an exercise for the reader.

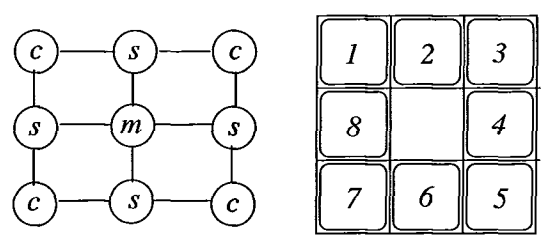

Figure 5: Side and Corner and Middle States in the Eight Puzzle.

In general, if we color the squares of a sliding-tile puzzle in a checkerboard pattern, the blank always moves from a square of one color to one of the other color. For example, in the Eight Puzzle, the s states will all be one color, and the rest will be the other color. If the two different sets of colored squares are entirely equivalent to each other, as in the five and fifteen puzzles, there will be a single branching factor at all levels. If the different colored sets of squares are different however, as in the Eight Puzzle, there will be different odd and even branching factors. In general, a rectangular sliding-tile puzzle will have a single branching factor if

at least one of its dimensions is even, and alternating branching factors if both dimensions are odd.

Application to FSM Pruning

So far, we have pruned duplicate nodes in the sliding-tile puzzle search trees by eliminating the inverse of the last operator applied. This pruning process can be represented and implemented by the finite state machine (fsm) shown in Figure 6. A node represents the last operator applied, and the arcs include all legal operators, except for the inverse of the last operator applied. Thus, the FSM gives the legal moves in the search space, and can be used to prune the search.

However, even more duplicates can be eliminated by the use of a more complex FSM. For example, there is cycle of twelve moves in the sliding-tile puzzles that comes from rotating the same three tiles in a two by two square pattern. Taylor and Korf (Taylor & Korf 1993) show how to automatically learn such duplicate patterns and express them in an FSM for pruning the search space. For example, they generate an FSM with 55,441 states for pruning duplicate nodes in the Fifteen Puzzle. An incremental learning strategy for FSM pruning is addressed by Edelkamp (Edelkamp 1997).

The techniques described here can be readily applied to determine the asymptotic branching factor of these pruned spaces. Since the number of different types of nodes is so large, only the numerical simulation method is practical for solving the resulting system of equations. For example, we computed a branching factor of 1.98 for the above mentioned FSM, after about 50 iterations of the recurrence relations. This compares with a branching factor of 2.13 for the Fifteen Puzzle with just inverse operators eliminated.

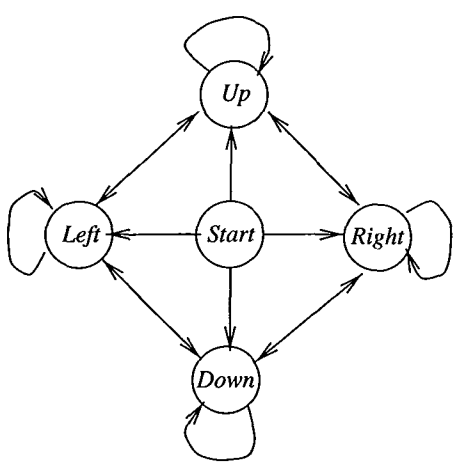

Figure 6: An automaton for predecessor elimination in the sliding tile puzzle.

Conclusions

We showed how to compute the asymptotic branching factors of search trees where different types of nodes have different numbers of children. We begin by writing a set of recurrence relations for the generation of the different node types. These recurrence relations can then be used to determine the exact number of nodes at a given depth of the search tree, in time linear in the depth. They can also be used to estimate the asymptotic branching factor very accurately. Alternatively, we can rewrite the set of recurrence relations as a set of simultaneous equations involving the relative frequencies of the different types of nodes. The number of equations is one greater than the number of different node types. For relatively small numbers of node types, we can solve these equations analytically, by finding the roots of the characteristic equation of a matrix, to derive the exact asymptotic branching factor. We give asymptotic branching factors for Rubik's Cube, the Five Puzzle, and the first ten square sliding-tile puzzles.

Acknowledgments S. Edelkamp is supported by DFG within graduate program on human and machine intelligence. R. Korf is supported by NSF grant IRI-9619447. Thanks to Eli Gafni and Elias Koutsoupias for helpful discussions concerning this research.

References

Edelkamp, S. 1997. Suffix tree automata in state space search. In *KI–97*, 381–385.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Motwani, R., and Raghavan, P. 1995. *Randomized Algorithms.* Cambridge University Press, Cambridge, UK.

Norris, J. R. 1997. *Markov Chains.* Cambridge University Press, Cambridge, UK.

Taylor, L. A., and Korf, R. E. 1993. Pruning duplicate nodes in depth-first search. In *AAAI–93*, 756–761.