How Long Will It Take? *

Ron Musick **Stuart Russell**
Computer Science Division
University of California
Berkeley, CA 94720, USA
musick@cs.berkeley.edu
russell@cs.berkeley.edu

Abstract

We present a method for approximating the expected number of steps required by a heuristic search algorithm to reach a goal from any initial state in a problem space. The method is based on a mapping from the original state space to an abstract space in which states are characterized only by a syntactic "distance" from the nearest goal. Modeling the search algorithm as a Markov process in the abstract space yields a simple system of equations for the solution time for each state. We derive some insight into the behavior of search algorithms by examining some closed form solutions for these equations; we also show that many problem spaces have a clearly delineated "easy zone", inside which problems are trivial and outside which problems are impossible. The theory is borne out by experiments with both Markov and non-Markov search algorithms. Our results also bear on recent experimental data suggesting that heuristic repair algorithms can solve large constraint satisfaction problems easily, given a preprocessor that generates a sufficiently good initial state.

Introduction

In the domain of heuristic problem solving, there is often little information on how many operations it will take on average for a particular algorithm to solve a class of problems. Most results in the field concern such things as dominance, worst case complexities, completeness and admissibility. Average-case analyses deal with individual algorithms, and few if any results have been obtained for search problems. Our approach is to directly address this issue by developing an approximation of the expected number of steps an algorithm requires to reach a solution from any initial point in the state space. This method avoids detailed analysis of the search algorithm, and has been used to generate reasonably accurate results for some small to medium sized problems; for some special cases, it is applicable to any size problem. The method is applicable to Markovian search algorithms such as hill-climbing, random-restart hill-climbing, heuristic repair methods, genetic algorithms and simulated annealing. With some additional work, it could be applied to any bounded-memory search algorithm.

We begin by introducing a mapping from a generic state space representation of any Markov process into a compact, abstract Markov model, in which states are characterized only by a syntactic "distance" from the nearest goal. This model is used to elicit a system of linear equations that can be solved by matrix methods to find the solution time for each state.

For some special cases of the system of equations, we derive closed form expressions that enable us to make quantitative statements about the solution time of an algorithm, given estimates of the likelihood that the algorithm can reduce the syntactic "distance" on any step. When the likelihood increases for states closer to a goal, the theory demonstrates the existence of an "easy zone" within some radius of the goal states. Within this radius the expected number of steps to solution is small, but once a narrow boundary is crossed the expected number of steps grows rapidly. This has some interesting implications on how the initial state of heuristic algorithms affects the average length of a successful solution.

The likelihood estimates can be obtained by theoretical analysis of the algorithm and problem space, or by sampling. Although these methods are beyond the scope of this paper, we show that our predictions are not generally oversensitive to errors in these estimates. Finally, we claim that this Markov model can be a *useful* (albeit not perfect) model of a non-Markov process. Some preliminary empirical results are offered in support of this claim.

An independent use of Markov models for algorithms appeared in in (Hansson *et al.*, 1990), which uses a different abstract space to optimize the search parameters of a hill-climbing algorithm. Work of a complementary nature has been done recently in (Cheeseman *et al.*, 1991). Their approach can be described as an empirical

*This research was supported in part by the National Science Foundation under grant number IRI-9058427

investigation of the relationship between the difficulty of solution of a (constraint satisfaction) problem and various syntactic characteristics of the problem. They also discover a sharp transition between easy and hard or unsolvable problems.

The Model

Our ability to find the expected number of steps to solution depends strongly on the representation of the problem.

A Typical State Space Representation

Any problem can be mapped into a state space representation consisting of a set S of states $\{S_0, S_1 \ldots, S_p\}$, a set $A \subset S \times S$ of weighted, directed arcs between the states, a set $G \subset S$ of the goal states, and a state $I \in S$. S is the set of possible states in the problem, A is the set of all transitions between states, G is the set of goal states, and I is the initial state (Newell and Simon, 1972). We assume WLOG:

- The weight on each arc is 1,
- Goal states are absorbing, meaning that there are no transitions out of a goal state,
- There are n features used to describe any state,
- Every operation, or transition, modifies only one feature in the current state, and therefore counts as one step. (Hence, each state has $\leq n$ outgoing arcs.)

When we apply, for example, a hillclimbing algorithm to a problem, we are walking a path from the state I to a state $g \in G$, where each step in the path is a transition from some state S_i to some state S_j, where $(S_i, S_j) \in A$.

The above is a fairly standard basis for problem solving in AI. When the algorithm to solve this problem is Markov[1] in nature, then it is possible to represent a problem *and* an associated algorithm by adding a probability to each transition. We therefore define p_{ij} to be the probability that the algorithm will move to S_j when in S_i.

Let $NS(S_i)$ be the expected number of steps to reach a solution from state S_i. To find the value of NS for all states we need only solve the the following set of linear equations:

For every $S_i \notin G$ and every $g_i \in G$,

$$\begin{aligned} NS(g_i) &= 0 \\ NS(S_i) &= \sum_{j=1}^{n} p_{ij} NS(S_j) + 1 \qquad (1) \end{aligned}$$

$NS(g_i) = 0$ because we are already at a goal state. The solution to this set of equations, if it exists, gives us the numbers we are looking for. There is, however, a major problem with such a direct approach. The state space of a small to medium sized problem is large enough to give us an enormous set of simultaneous equations, far too many to solve. Consider the 50-queens problem where the goal is to place 50 queens on a 50x50 chess board without them attacking each other. In the most natural formulation of this problem, there are 50 variables, 1 for each row of the chess board. There are 50 possible values per variable. That leads to a state space with 50^{50} states, and therefore 50^{50} simultaneous equations to solve! This is not very useful.

Change of representation is a powerful tool in such cases, and has been used in the past to make classes of problems more tractable (Amarel, 1981; Nadel, 1990). Below we introduce a change of representation that drastically reduces the number of states in our state space representation, while retaining the information needed to estimate solution times. This change of representation is general enough to be applicable any problem/algorithm pair that can be modeled as above.

Our Representation

Let DIF_i be the set $\{d_{i1}, \ldots, d_{im}\}$ such that for state S_i, d_{ij} is the number of features of S_i that are different from the goal state $g_j \in G$. Define the distance $dist(S_i)$ to be $min(DIF_i)$. With this definition of distance, every goal state is at a distance of 0, and the largest distance possible is n, the number of features. Also, the shortest ***path*** from a state at a distance of i to the closest goal state will be $\geq i$ in length. Note that with this definition, a given transition can decrease the distance while increasing the length of the shortest solution path.

The new representation will have $n + 1$ abstract states $\{D_0, ..., D_n\}$, where D_i corresponds to the set of states at a distance of i. The new goal state D_0, corresponding to the set of goal states G, is the only absorbing state. The states in this model no longer form a state space for the original problem/algorithm pair, but instead for the Markov model that approximates it. Since we allow transitions to affect only 1 feature, a transition from D_i will leave us in either D_{i-1}, D_i, or D_{i+1}. Thus there are a maximum of three transitions out of every state. The new transition probabilities $tp_{ii}, tp_{i,i-1}, tp_{i,i+1}$, which must of course sum to 1, are defined as:

$$tp_{ij} = \textstyle\sum_{S_k \in D_i} (P(S_k) \sum_{S_m \in D_j} p_{km})$$

$$P(S_k) = \textstyle\sum_{S_j} P(S_j) p_{jk} / \sum_{S_j, S_l | S_l, S_k \in D_i} p_{jl}$$

where, given that the dist$(S_k) = i$ and that we are currently in some state with a distance of i, $P(S_k)$ is the prior probability that the state we are in is S_k.

These equations show that it is possible, in principle, to derive the new transition probabilities from the transition probabilities in the original space. However, it appears from the sensitivity analysis that it will also be acceptable (and often much more practical) to estimate these probabilities *without any knowledge* of the

[1]The decision of which action to take in any state depends only on the current state and the set of transitions out of this state; there can be no direct dependence on past states.

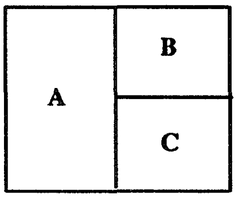

Figure 1: **A Simple 3-Coloring Problem**
Regions A, B and C are to be colored red (r), green (g) or blue (b).

State	Feature Values	Transition Probabilities	Goal?	Initial?
S_0	r,r,r	$\frac{1}{6}S_1, \frac{1}{6}S_2, \frac{1}{6}S_3$ $\frac{1}{6}S_6, \frac{1}{6}S_9, \frac{1}{6}S_{18}$	no	no
S_1	r,r,g	$\frac{1}{6}S_0 \dots \frac{1}{6}S_{19}$	no	no
S_2	r,r,b	$\frac{1}{6}S_0 \dots \frac{1}{6}S_{20}$	no	no
S_3	r,g,r	$\frac{1}{6}S_4 \dots \frac{1}{6}S_{21}$	no	no
S_4	r,g,g	$\frac{1}{6}S_3 \dots \frac{1}{6}S_{22}$	no	yes
S_5	r,g,b	S_5	yes	no
⋮	⋮	⋮	⋮	⋮
S_{25}	b,b,g	$\frac{1}{6}S_{24} \dots \frac{1}{6}S_{16}$	no	no
S_{26}	b,b,b	$\frac{1}{6}S_{24} \dots \frac{1}{6}S_{17}$	no	no

Table 1: **Initial Representation of Problem**
The feature values shown are the colors associated with regions A, B and C, respectively. The transition probabilities show that our algorithm randomly selects a feature to change, then randomly changes the value.

original transition probabilities, by sampling the algorithm's behavior on a set of problems. Results of both methods are illustrated below.

We now give a small example to demonstrate the transformation. Consider the 3-coloring problem in Figure 1, where the possible colors are $\{r, g, b\}$ (red, green, blue). The goal is to find a color for each region so that there are no neighboring regions with the same color. The allowable operations modify the color of one region at a time; thus there are a total of six possible transitions from any non-goal state. Let the algorithm for this problem be random, so that from the current state, one of the three regions is randomly chosen, then its color is changed to one of the two remaining colors different from its current color. Table 1 shows the original state space representation, Table 2 shows our representation.

State	Represents States	Transition Probabilities	Goal?	Initial?
D_0	$S_{5,7,11,15,19,21}$	D_0	yes	no
D_1	$S - D_0 - D_2$	$\frac{1}{3}D_0, \frac{1}{2}D_1, \frac{1}{6}D_2$	no	yes
D_2	$S_{0,13,26}$	D_1	no	no
D_3	{}	{}	no	no

Table 2: **Final Representation of Problem**
The second column shows where the states of the original representation have been mapped.

From the information in Table 2, we can predict a very quick solution. The transition probabilities tell us that from state D_2 there is no choice but to move closer to a solution, to D_1. From state D_1, we have twice the chance to move to the goal state D_0 than to move further away again to D_2. We will show below that, as one would imagine, the expected number of steps to solution is very dependent on the relative probabilities of moving towards or away from a solution. In fact, we explicitly measure the effect of this ratio on the expected number of steps for some special cases.

The main result of this new representation is that the system of equations (1) becomes drastically smaller and simpler. For the general state space representation the size drops from $O(n^k)$ to $O(n)$ where n is the number of features, and k is the magnitude of the domain of the features.

$$\begin{aligned} NS(D_0) &= 0 \\ NS(D_i) &= tp_{i,i-1}NS(D_{i-1}) + tp_{i,i}NS(D_i) + \\ &\quad tp_{i,i+1}NS(D_{i+1}) + 1 \end{aligned} \tag{2}$$

Theoretical Implications of the Model

We can rewrite the system of equations (2) as:

$$\begin{aligned} NS(D_0) &= 0 \\ -tp_{i,i-1}NS(D_{i-1}) + & \\ (1 - tp_{i,i})NS(D_i) - tp_{i,i+1}NS(D_{i+1}) &= 1 \end{aligned} \tag{3}$$

Then putting this in matrix form,

$$\mathbf{M} \times \mathbf{NS} = \mathbf{F} \tag{4}$$

where $\mathbf{M}$ is an $(n+1) \times (n+1)$ coefficient matrix, $\mathbf{NS}$ is the column vector $(NS(D_0), \dots, NS(D_n))^T$, and $\mathbf{F}$ is the column vector $(0, 1, \dots, 1)^T$. By nature, $\mathbf{M}$ is a *tridiagonal* matrix, a matrix with 0 entries everywhere but the diagonal and the two bands around it. On the lower band, all entries are of the form $-tp_{i,i-1}$. These represent the (negative) probabilities of moving closer to the goal, from D_i to D_{i-1}. On the upper band, all the entries have the form $-tp_{i,i+1}$. These represent the probabilities of moving away from the goal. The diagonal contains entries of the form $(1 - tp_{ii}) = tp_{i,i-1} + tp_{i,i+1}$. For convenience sake, throughout the rest of the paper we will call the lower band $\mathbf{A}$ (= $[a_i]$, an $n \times 1$ column vector), the upper band $\mathbf{C}$ (= $[c_i]$) and the diagonal $\mathbf{B}$. Thus a 4×4 matrix $\mathbf{M}$ will look like:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 \\ 0 & a_2 & b_2 & c_2 \\ 0 & 0 & a_3 & b_3 \end{pmatrix} \tag{5}$$

where in general $c_n = 0$. In Figure 2 we show an example[2] of how a_i, b_i and c_i can vary as a function of the distance i.

[2]The particular transition probabilities shown were derived from mathematical analysis of a heuristic repair algorithm (Minton *et al.*, 1990) applied to the class of random constraint satisfaction problems. This application is discussed further below.

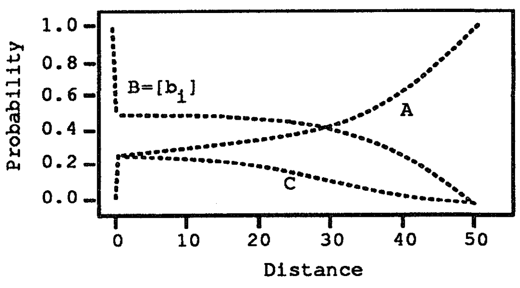

Figure 2: **Transition Probability Vectors**
The vectors **A**, **B** and **C** as a function of distance. For example, $-tp_{10,9} = A_{10} = [a_{10}] \approx .31$ is the probability of moving from D_{10} to D_9 on the next step.

Because **M** is a tridiagonal matrix, solving for $NS(D_i)$ is a very efficient $O(n)$ process. Furthermore, for some special cases of the **M** matrix, we can get a *closed form* for $NS(D_i)$. This not only allows us to calculate the solution time of a particular problem immediately, but, more importantly, it also tells us exactly what is influencing the expected number of steps over a whole class of problems.

Closed Forms

In this section we explore the case where $a_i = a_j = a$ and $c_i = c_j = c$, meaning that the **A**, **B** and **C** probability vectors are horizontal lines. This is not realistic for most problems, but it is an interesting case to examine (a *biased random walk*, or, in physics, a random walk in a uniform field). Solving the system of equations (4) for closed forms gives us:

For $tp_{1,0} = -a = -c$,

$$NS(D_i) = \frac{(2n - i + 1)i}{2tp_{1,0}} \tag{6}$$

For $c = \phi a$, or $tp_{1,2} = \phi tp_{1,0}$ and $\phi \neq 1$

$$NS(D_i) = \frac{\phi^{n+1}(1 - \phi^{-i}) + i(1 - \phi)}{tp_{1,0}(1 - \phi)^2} \tag{7}$$

The correctness of these equations has been tested empirically by comparing their predictions to those generated by solving the system of equations (4). The test set consisted of about 50 different problem/algorithm descriptions, each of size $n = 50$. Note that we can derive equation (6) from (7) by letting $\phi = 1$ and applying L'Hôpital's rule twice.

Based on equations (6) and (7) we can make several statements about the characteristics of the solution. From equation (6) we can see that when a = c:

- $NS(D_i)$ is monotonically increasing as the distance i increases from 0 to n. (This also holds for equation (7).)
- $NS(D_i)$ is directly proportional to $\frac{1}{b}$. This follows from $a = c$ and $b = -a - c = 2tp_{1,0}$. As the tendency to stay the same distance from the goal (tp_{ii}) increases, so increases the expected time to solution.

For the case where $c = \phi a$, equation (7) gives us some interesting information:

- For $\phi < 1$, or when the probability to move closer to the goal is greater than to move away, the terms ϕ^{n+1} and ϕ^{n+1-i} are small, and the dominant term is $i(1 - \phi)$. This shows that for this case, $NS(D_i)$ increases nearly linearly as the distance from solution grows, indicating that the problem will be easy to solve no matter what the initial state is. This coincided with the sample runs we have done.
- For $\phi > 1$, or when we are more likely to move away from the goal than towards it, the terms ϕ^{n+1} and ϕ^{n+1-i} are large, positive and dominant. Furthermore note that within a very short distance i from the goal state D_0, $NS(D_i)$ will grow to nearly its maximum value (see curve LA in Figure 3). This tendency increases as ϕ grows larger. This is because at $i = 1$ the dominant term is $\phi^{n+1} - \phi^n \approx \phi^{n+1}$. As i increase to n, the dominant term grows monotonically until it reaches ϕ^{n+1}. This early jump suggests that when $\phi > 1$, if we do not start at a solution then we will never find a solution.

Solutions for other continuous functions

The values in the **A** and **C** column vectors in the previous section were restricted so that $a_i = a_j$ and $c_i = c_j$. In this section we relax that restriction, and allow the values in **A** and **C** to vary according to any general function. Of course, the sum $a_i + b_i + c_i = 0$ must still be true, by definition. Unfortunately, even for the case of linear variation of the values of **A** and **C**, we were unable to come up with a closed form. In fact, we believe it to be impossible to do so.

We can still resort to solving the set of equations in (4) to get a feeling for $NS(D_i)$ in these more complicated systems. In Figure 3, we show the log plot of $NS(D_i)$ for several different problem/algorithm descriptions generated by solving the system of equations (4). From these plots and others like them, we have the following observations to make:

- Our solutions have an interesting new characteristic. From the horizontal line cases described by equations (6) and (7), we saw that the largest rate of increase occurs at distances near 0. However, when looking at lines LB and LC in Figure 3, we see a region between $0 \leq i \leq 16$ in which the expected number of steps to solution is small. Outside the region, the expected number of steps to solution grows rapidly again. For example, for case LC in Figure 3, $NS(D_i)$ explodes at about 17, doubling with each step until reaching 23. For case LB the growth is even more dramatic.
- The boundary between this easy zone and the hard region can be very sharp, spanning just a few steps.
- All of the hard problems reach asymptotes very quickly. The reason for this is that from any point, the probability of reaching state D_n before state D_0

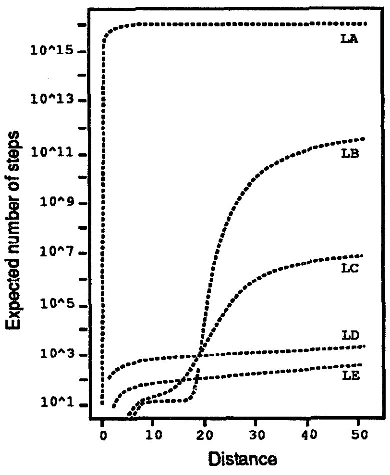

Figure 3: **Log plot of Expected Solution Times**
Log plots of expected solution times as a function of the distance of the initial state for several different problem/algorithms.
LA: **A**, **B** and **C** are horizontal lines, $c = 2a$.
LB: **A** is a cosine curve $A_1 \approx 1, A_{49} \approx 0, C = 1 - bfA$.
LC: **A** is a line, $A_1 \approx 1, A_{49} \approx .2$, **C** $= 1 -$ **A**.
LD: **A**, **B** and **C** are horizontal lines, **C** $=$ **A**.
LE: **A**, **B** and **C** are as depicted in Figure 2.

is near 1, and the expected length of the solution from state D_n is a constant.

This shows that for any heuristic problem solver, the choice of an initial state has a heavy impact on the expected solution time for the problem. Consider heuristic repair algorithms like that in (Minton *et al.*, 1990). If indeed the preprocessor does not choose an initial state close to a goal state, then the algorithm will fail with high probability. A closed form solution will allow us to quantify the size of the easy zone, the sharpness of the boundary, and the magnitude of the subsequent jump of $NS(D_i)$.

Sensitivity

One issue that has not yet been addressed involves the sensitivity of the solution to errors in the estimates of **A**, **B** and **C** when we resort to solving the system of equations (4). After all, it might be unreasonable to expect to be able to determine the transition probabilities to within tenths of a percent, or even several percent of the true values.

Let's formalize this notion. Looking back at (3) and (4) we remember that **F** is a known matrix, but **M** is a matrix of error prone functions of transition probabilities. Thus, we let **TNS** be the true $NS(D_i)$, **TM** be the true coefficient values, and examine the effects of adding an error matrix **X** to the true coefficient values **TM**. This type of analysis is presented in (Golub and Van Loan, 1983; Lancaster and Tismenetsky, 1985).

$$(\mathbf{TM})\mathbf{TNS} = \mathbf{F} \quad (8)$$
$$(\mathbf{TM}+\mathbf{X})\mathbf{NS} = \mathbf{F}$$
$$\mathbf{NS} = (\mathbf{TM}+\mathbf{X})^{-1}\mathbf{F} \quad (9)$$

We use the first few terms of a Taylor series expansion to get:

$$(\mathbf{TM}+\mathbf{X})^{-1}\mathbf{F} \doteq \mathbf{TM}^{-1}(\mathbf{X})\mathbf{TM}^{-1}\mathbf{F} + \mathbf{TM}^{-1}\mathbf{F} \quad (10)$$

Merging (8), (9) and (10) we get

$$\mathbf{NS} \doteq \mathbf{TNS} + \mathbf{TM}^{-1}(\mathbf{X})\mathbf{TM}^{-1}\mathbf{F} \quad (11)$$

Our measure of sensitivity is the relative error in **TNS**, or $\frac{||\mathbf{TNS}-\mathbf{NS}||}{||\mathbf{TNS}||}$, calculated as:

$$\frac{\mathbf{TM}^{-1}(\mathbf{X})\mathbf{TM}^{-1}\mathbf{F}}{\mathbf{TM}^{-1}\mathbf{F}} \leq \kappa(\mathbf{TM})\frac{||\mathbf{X}||}{||\mathbf{TM}||} \quad (12)$$

where $\kappa(\mathbf{TM}) = ||\mathbf{TM}||\ ||\mathbf{TM}^{-1}||$ is the condition number of the **TM** matrix, and $\frac{||\mathbf{X}||}{||\mathbf{TM}||}$ is the relative error of **TM**.

We can, of course, construct situations in which the **TM** matrix is very poorly conditioned. For example, if we let tp_{ii} be very close to 1, so that the probability of moving off state i is practically nil, then the corresponding $b_i \approx 0$, and thus $||\mathbf{TM}^{-1}||$ will be very large. In more normal situations, however, the **TM** matrix is well conditioned. Experimentally, perturbing the **M** matrix in (4) has led to similar relative disturbances in $NS(D_i)$. This is good; it implies that our confidence in $NS(D_i)$ can be nearly as high as our confidence in the **M** matrix that we created.

Markov Models

In general, the abstraction mapping we have applied does not preserve the Markov property of the original algorithm, and presumably the approximation will be even greater for an algorithm that is not Markov in the original space. At present, we can offer only empirical evidence that the results we obtain in our simple model bear a reasonable resemblance to the actual performance of the algorithm. Given our aims, it may be acceptable to lose an order of magnitude or two of accuracy in order to get a feeling for complexity of the given problem/algorithm pair.

To investigate this issue, we built a non-Markov algorithm for the 8-puzzle problem and modeled it using the techniques described in the paper. The algorithm used to solve the 8-puzzle is a 3-level look ahead hill climbing search guided by a modified manhattan distance heuristic. The modification involves a penalty applied to any move that would place us in a state we have recently visited. When there are several options that look equivalent in the eyes of this heuristic, we choose randomly.

Using this algorithm, we solved 100 randomly generated 8-puzzles from each starting distance of 3, 4,

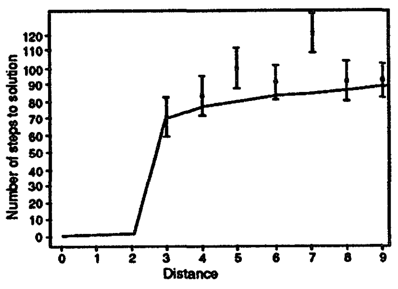

Figure 4: **A Markov model of a non-Markov process**
The line shown is the expected number of steps to solution as predicted by the theory. The cross-bars show the mean and deviation of 100 actual runs from each initial distance.

5, 6, 7, 8 and 9. For each run we kept track of the number of steps required to reach the solution. If the number exceeded 1000, we discarded the run. For each set of runs, we calculated the sample mean and the deviation of the sample mean. These numbers represent the actual solution length characteristics of the non-Markov process. To generate the Markov model, we again solved 100 randomly generated 8-puzzles from each starting distance. These runs were used to calculate the transition probabilities by sampling, as described earlier. Using these transition probabilities, we solved the system of equations (4) to find $NS(D_i)$. Figure 4 shows the results. As can be seen from the figure, the resulting accuracy is quite acceptable.

Conclusion

This work represents the early stages of an effort to get a better understanding of how to solve combinatorial problems. The logical conclusion of the effort will be an effective method for estimating the complexity (and variation thereof) of solving a member of a given class of problems, based on some description of the class. Results in this paper and in (Cheeseman *et al.*, 1991) show that the complexity can be highly dependent on certain parameters of the problem, and we have shown that these parameters can be estimated reasonably well. An understanding of this extreme complexity variation would be very useful for any system that uses metareasoning to control combinatorial problem-solving, to allocate effort among subproblems, or to decide how much effort to put into preprocessing.

The existence of an "easy zone" suggests that successful application of heuristic repair methods, which begin with an incorrect assignment of variables and gradually modify it towards a consistent solution, will depend on the density of solution states, the accuracy of the heuristic, and the quality of the preprocessed starting state. In a forthcoming paper, we derive these quantities for random constraint satisfaction problems, and examine the correspondence of our model to actual performance.

Acknowledgements

We would like to thank Jim Demmel and Alan Edelman for their excellent suggestions dealing with matrix theory.

References

Amarel, S. 1981. On representation of problems of reasoning about actions. In Webber, B. L. and Nilsson, N. J., editors 1981, *Readings in Artificial Intelligence.* Morgan-Kaufmann, Los Altos, California.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *Proceedings of the Twelfth International Conference on AI.* 331–337.

Golub, G. H. and Van Loan, C. F. 1983. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Maryland.

Hansson, O.; Holt, G.; and Mayer, A. 1990. Toward the modeling, evaluation and optimization of search algorithms. In Brown, D. E. and White, C. C., editors 1990, *Operations Research and Artificial Intelligence: the Integration of Problem Solving Strategies.* Kluwer Academic Publishers, Boston.

Lancaster, P. and Tismenetsky, M. 1985. *The Theory of Matrices, Second Edition.* Academic Press, New York.

Minton, S.; Johnston, M.; Philips, A.; and Laird, P. 1990. Solving large-scale constraint satisfaction and sceduling problems using a heuristic repair method. In *Proceedings of the Eigth National Conference on AI.* 17–24.

Nadel, Bernard A. 1990. Representation selection for constraint satisfaction: A case study using n-queens. *IEEE Expert* 5(3):16–23.

Newell, A. and Simon, H. A. 1972. *Human Problem Solving.* Prentice hall, Englewood Cliffs, New Jersey.