Contents

1 Introduction 2

2 Background 4
   2.1 Backtracking algorithms ........................................... 4
   2.2 Computing the success probability of consistency checks .......... 6
   2.3 The model for backtracking by Haralick and Elliott ................ 7
   2.4 Difference in costs between first solution and average solution .... 9

3 A new model for standard backtracking, first solution 11
   3.1 Success and failure probability when searching ..................... 11
   3.2 Average number of consistency checks ............................... 15
   3.3 A computer program .................................................. 17

4 A new model for min forward checking, first solution 18
   4.1 Adaptation of the immediate parameters ............................. 18
   4.2 Adaptation of the level equations ................................... 21
   4.3 A computer program .................................................. 21

5 Model predictions 22
   5.1 Modelling standard backtracking: first versus all solutions ........ 22
   5.2 Model predictions for the average costs ............................. 23

6 Experimental 26
   6.1 Predicting the behavior of random CSP problems .................... 26
   6.2 Predicting the behavior of the N-queens problem .................... 29
   6.3 Predicting phase transitions for graph coloring problems ............ 31

7 Conclusions, expectations, and future research 34

References 35
Predicting the behavior of search algorithms on CSP problems (cont’d)

Gidi van Liempd
KPN Research
P.O. Box 15000
9700 CD Groningen, The Netherlands
E.P.M.vanLiempd@research.ptt.nl

Jaap van den Herik
University of Limburg
P.O. Box 616
6200 MD Maastricht, The Netherlands
herik@cs.rulimburg.nl

April 24, 1995

Abstract: We have designed and implemented two new models for backtracking algorithms on Constraint Satisfaction Problems (CSP). The models are based on the probability that a consistency check succeeds, independent of the variables and values involved and independent of any past processing. In this article we investigate whether such models can be used to predict the average number of consistency checks needed to find a first solution to a given CSP problem. First, we formulate a new model for standard backtracking. Then, we formulate an extension of the model for min forward checking. We have implemented both models and performed experiments on random CSP problems, and on instances of non-random problems, such as the N-queens problem. Moreover, we have applied the model for standard backtracking to predicting the phase transition for graph coloring problems. Our conclusion is that the new models are viable. In addition, we conclude that for random CSP problems we are able to predict, with reasonable accuracy, the probability of finding a solution and its average cost. From this result it follows that our models can be used when selecting a search algorithm which we would like to be efficient for a given CSP problem.

*The research reported in this contribution was carried out in the framework of the EXPLOIT project, a joint project of KPN Research and the University of Limburg. The author has a second affiliation as a PhD student at the Department of Computer Science of the University of Limburg, Maastricht, The Netherlands.
1 Introduction

The effort involved in finding a solution for a given Constraint Satisfaction Problem (CSP) depends heavily on the search algorithm employed. The goal of our research is predicting the behavior of CSP search algorithms for a given CSP representation of a problem. In particular, we are interested in predicting the cost of finding a first solution, and the probability that we will find such a solution. If we can predict (accurately) the cost of finding a solution given a particular search algorithm, these predictions may help us in selecting the most suitable search algorithm for any problem at hand.

To achieve our goal, we try to capture the essential characteristics of the CSP representation and the search algorithm in a model, and then use that model to make predictions about the expected behavior. In this article we describe a model in which the problem characteristics are abstracted into three parameters only: the number of variables in the CSP problem, the number of values each variable may have, and the probability that a consistency check between a given pair of variables with values assigned will succeed. We investigate whether such a model is sufficient for our purpose.

Our model is based on the probability $p$ that a consistency check succeeds, independent of the variables and values involved in the checking, and independent of any past processing. This probability is introduced by Haralick and Elliott [6], who used it to predict the cost of backtracking algorithms for finding all solutions of a given CSP problem. Our research can thus be considered an extension of their work to the case of finding a first solution.

There are a few other statistical models for backtracking, e.g., Stone and Sipala [16] and Pearl [12], but these do not particularly aim at CSP problems. Stone and Sipala [16] used a model of searching in a binary tree with a fixed cutoff probability. Assuming that this fixed cutoff probability is associated with the probability of failure of a value assignment in a CSP problem, it is clear that the model does not suffice since the latter probability increases when going more deeply into the search tree. Pearl [12] described two models for backtracking. The first model assumes either a cost zero or one for a branch in the search tree, and is therefore not suitable: when using backtracking on a CSP problem, the cost (number of consistency checks) associated with a value assignment (branch) increases as we go more deeply into the search tree. The second model is not suitable since it is based on the properties of a heuristic evaluation function in a format that is not used in CSP representations.

Our research can be considered as an elaboration of a suggestion by Nudel [11]. He generalized Haralick and Elliott’s model by introducing a parameter, called inter-variable compatibility, to characterize CSP problems. Carrying out expected-complexity analyses for the problem of finding all solutions, he conjectured that this parameter might also be useful for the problem of finding one solution for a given CSP problem. In Nudel’s terminology, the current article aims at providing a level-0 theory for the latter problem.

Another close relation is to Gaschnig’s research ([4], Ch. 4). He performed measurements on various classes of CSP problems and introduced a degree of constraint, called $L$, identifying different CSP problems as parametrically equivalent. ($L$ stands for link percentage.) His

---

1This increase in failure probability was shown by Stone and Stone [17] for the $N$-queens problem.
research question was whether finding a first solution for similar problems required, on the average, equal costs. Gaschnig made a large amount of observations, but did not arrive at a definite model. In this respect we continue his research, and show that statistical models can be constructed describing the average search behavior for different parameter values.

Researchers such as Smith [14, 15], and Williams and Hogg [19, 20, 21] have also used (simple) models to predict other elements of CSP search behavior, viz. the location of the phase transition and the costs of finding a solution in that region. However, when selecting a search algorithm for finding a solution for a problem, the phase transition is not the only region of interest.

In section 2, we provide some necessary technical background information. Then, in sections 3 and 4 we present our two new models predicting the average number of consistency checks needed to find a first solution for a given CSP problem, when using standard back-tracking and min forward checking, respectively. In section 5, we describe the relation of our model predictions with other models. There we show for both models that the predictions for average costs and probability of finding a solution conform to experimental results on random CSP problems. Section 6 provides experimental results for the N-queens problem and the graph coloring problem. It also discusses the outcomes. In section 7, we draw some conclusions and hint at future research.
2 Background

In this section we provide the necessary background for predicting the average number of consistency checks needed to find a first solution for a given CSP problem. We start with a definition of a CSP problem and its constraints. Subsequently we describe the algorithms standard backtracking and min forward checking. Then we show how the probability of the success of a consistency check is related to the number of constraints. After the presentation of the model by Haralick and Elliott [6] for finding all solutions, we clarify the difference in costs between a first solution and an average solution.

Ginsberg [5] defines a CSP problem as follows:

A constraint satisfaction problem (CSP) is defined by the triplet \((I, V, \kappa)\). \(I\) is a set of variables; for each \(i \in I\), there is a set \(V_i\) of possible values for the variable \(i\). \(\kappa\) is a set of constraints; each pair \((J, P) \in \kappa\) is called a constraint with \(J = (j_1, \ldots, j_k)\) an ordered subset of \(I\), and \(P\) a subset of \(V_{j_1} \times \cdots \times V_{j_k}\). A solution to the CSP problem is a set \(v\) of values for each of the variables in \(I\) such that \(v_i \in V_i\) for each \(i\) and for every constraint \((J, P)\) of the above form in \(\kappa\), \((v_{j_1}, \ldots, v_{j_k}) \in P\) holds.

We restrict our attention to CSP representations with \(N\) variables \((\|I\| = N)\) each having a domain of \(M\) values \((\|V_1\| = \cdots = \|V_N\| = M)\). Moreover, we limit the constraints to binary constraints, i.e., related to two (different) variables at a time. In a solution, each variable has only one value. Therefore, any binary constraint can be represented by a set of binary constraint pairs of the form \([(i, v_i), (j, v_j)]\). Such a constraint pair would indicate that value \(v_i\) for variable \(i\) is compatible with value \(v_j\) for variable \(j\). A consistency check is the basic operation in CSP problems, verifying whether the representation explicitly permits a given constraint pair. We measure the cost of solving a CSP problem in terms of these consistency checks.

2.1 Backtracking algorithms

Many CSP search algorithms assign values to variables sequentially. At any time in the search process, it is true that some variables have values assigned to them, and others have not. The former are called past variables, the latter future variables (see, e.g., Meseguer [8]). The basic step in these algorithms consists of selecting a current variable from the future variables for assigning a value to it. Then, the algorithm tests the value assignment. Details of the test depend on the algorithm used, but always involve consistency checks, e.g., of the new value against the actual values of past variables or against the possible values of future variables, or even against some possible values of past variables. Whatever the details, if the test succeeds, a new current variable is selected or a solution has been found; if the test fails, a new value assignment must be made.
Below, we describe a standard frame for the class of **backtracking algorithms** in pseudo code.

```pseudo
backtrack_search (past_vars, future_vars)
1. IF solution found /* all variables have compatible values */
2. THEN
3. present solution
4. return from this call reporting SUCCESS.
5. ELSE
6. select current_var from future_vars
7. future_vars' := future_vars - current_var
8. past_vars' := past_vars + current_var
9. WHILE solution not found
10. AND current_var has untried possible values left
11. DO
12. select value from possible values for current_var
13. IF value compatible with values of past_vars
14. AND value meets possible other criteria
15. THEN
16. call backtrack_search (past_vars', future_vars')
17. ENDIF
18. ENDDO
19. IF solution not found
20. THEN
21. return from this call reporting FAILURE.
22. ENDIF
23. ENDIF
```

The pseudo-code lines 6, 12, and 14 allow distinct variants within this class, viz. by selecting a variable, selecting a value, and applying other criteria, respectively. Below, we provide three variants. We start by defining standard backtracking:

<table>
<thead>
<tr>
<th>select variable</th>
<th>lexicographically</th>
</tr>
</thead>
<tbody>
<tr>
<td>select value</td>
<td>lexicographically</td>
</tr>
<tr>
<td>other criteria</td>
<td>none</td>
</tr>
</tbody>
</table>

The standard-backtracking algorithm uses a fixed ordering for selecting the current variable, and for selecting a value for each current variable. The value of the current variable is checked for consistency against the values of the past variables. If any of these checks fails, the algorithm takes the next untried value. If all values have been tried and failed, the algorithm resets the current variable and returns (backtracks) to the previous current variable, of which the current value will be retracted. Then, for this variable, the next untried value will be selected, and the algorithm continues as before. For this algorithm we have developed a new model in section 3.

<table>
<thead>
<tr>
<th>select variable</th>
<th>lexicographically</th>
</tr>
</thead>
<tbody>
<tr>
<td>select value</td>
<td>lexicographically from remaining values</td>
</tr>
<tr>
<td>other criteria</td>
<td>compatible values must be available for future variables</td>
</tr>
</tbody>
</table>
As in standard backtracking, we define forward checking [6] by instantiating the lines 6, 12, and 14. The forward-checking algorithm is basically the same as standard backtracking, except that it additionally checks whether all future variables have at least one possible value compatible with the current values of the past variables.

If we check the value of the current variable against all remaining values of the future variables (remaining values are values that are still possible with respect to past variables), and we update the remaining values for the backtrack search at the next level (line 16 in our algorithm), we have trivially fulfilled line 13 at this next level. This is the usual implementation of forward checking (see, e.g., [6, 11]).

**min forward checking**

<table>
<thead>
<tr>
<th>select variable</th>
<th>with least number of remaining possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>select value</td>
<td>lexicographically from remaining values</td>
</tr>
<tr>
<td>other criteria</td>
<td>compatible values must be available for future variables</td>
</tr>
</tbody>
</table>

The algorithm min forward checking [16] is an extension of forward checking with a search rearrangement heuristic. In this algorithm, after forward checking has been successful for a value of the current variable, a future variable with the least number of remaining possible values is selected as the next current variable. The algorithm is more efficient than standard backtracking for a significant class of problems [7]. For instance, for the $N$-queens problem and its variants it is a very efficient algorithm [17], although there exist even better algorithms (e.g., see [9]). When applying the min forward checking algorithm to a given CSP problem, we aim at predicting its behavior. Hence, we have developed two models for predicting the average cost of a first solution: first for standard backtracking, and thereafter for min forward checking.

### 2.2 Computing the success probability of consistency checks

In our models, we assume that the probability $p$ of a successful consistency check is independent of the variables and values involved, and also independent of any past processing. In this subsection we show how the probability, under these assumptions, is related to the number of constraint pairs in a given CSP representation.

Assume a CSP representation of a problem consists of $N$ variables each having domains of $M$ values, and binary constraints as described above. The total number of possible binary constraint pairs is

$$\binom{N}{2}M^2$$

(1)

(Constraint pairs $[(\text{Var}_X, \text{Val}_A),(\text{Var}_Y, \text{Val}_B)]$ and $[(\text{Var}_Y, \text{Val}_B),(\text{Var}_X, \text{Val}_A)]$ are considered the same).

When encoding a given problem into a CSP representation, we determine the total number of available binary constraint pairs, which we call $B$. The question now is, how to relate $p$ to $B$. If we randomly select two distinct variables each having a value assigned, the probability $p$ that those two values are compatible equals the probability of selecting an available binary constraint pair from the set of all possible binary constraint pairs:
\[ p = \frac{B}{\binom{N}{2}M^2} = \frac{2B}{N(N-1)M^2} \]  
\hspace{1cm} (2)

Equation (2) gives the important relation between \( p \) and the number of available binary constraint pairs.

A method of characterizing (random) CSP problems other than by the number of binary constraint pairs is given in Smith [14, 15] and Prosser [13]. In addition to the number of variables, \( N \), each with a set of \( M \) values, they use two parameters to describe the constraints. The \textit{constraint density} \( p_1 \) means the probability that a constraint between a given pair of variables is present. The \textit{constraint tightness} \( p_2 \) is a conditional probability that a pair of values is not compatible for a pair of variables, given that there is a constraint between the variables. These definitions lead to equation (3).

\[ B = p_1 \left( \frac{N}{2} \right) (1 - p_2)M^2 + (1 - p_1) \left( \frac{N}{2} \right) M^2 \]  
\hspace{1cm} (3)

In words: there are \( \binom{N}{2} \) variable pairs. Of these pairs, a fraction \( p_1 \) will have constraints between them. For each variable pair, there are \( M^2 \) possible value pairs. If the variable pair has a constraint, a fraction \( p_2 \) of these value pairs will be considered incompatible. Hence, for the variable pairs which have a constraint, a fraction \( (1 - p_2) \) of the value pairs will still be available. For the variable pairs that do not have a constraint, all \( M^2 \) value pairs are available.

Substituting \( B \) (from equation (3)) into equation 2 yields

\[ p = p_1 (1 - p_2) + (1 - p_1) = 1 - p_1 p_2 \]  
\hspace{1cm} (4)

This equation will allow us to check whether the predictions by our model are consistent with the observations by Smith and Prosser.

2.3 The model for backtracking by Haralick and Elliott

In 1980, Haralick and Elliott [6] presented a statistical model for the \textit{backtracking} algorithm in a CSP representation, as well as a variation of this model for the \textit{forward-checking} algorithm. Below, we recapitulate the model for backtracking, and use it as a basis for our model in section 3.

Haralick and Elliott grounded their model on the assumption that any two values, assigned to two distinct variables, are compatible with probability \( p \); \( p \) being independent of variable, value, and past processing. They assumed a CSP representation with \( N \) variables, each having \( M \) possible values. Any \( k \)-tuple of values for any \( k \) variables must satisfy \( \frac{1}{2}k(k-1) \) consistency checks among these variables. Given the assumptions for \( p \), the \( k \)-tuple of values therefore has a probability of \( p^{k(k-1)/2} \) of being consistent. There are \( M^k \) possible value assignments for \( k \) variables, hence the average\(^2\) number of consistent value assignments for \( k \) variables is

\[^2\text{The original article [6] uses here \textit{expected} instead of \textit{average}. For normal distributions, these coincide.}\]
In standard backtracking, a node in the search tree at level $k - 1$ will only be expanded into nodes at level $k$ if it is consistent at level $k - 1$. Since each variable can have $M$ values, the successful nodes of level $k - 1$ (being $M^k p^{(k-1)(k-2)/2}$) will be expanded into a total of $M^k p^{(k-1)(k-2)/2}$ tree search nodes at level $k$.

Each node at level $k$ in the search tree represents a value assignment for the $k$-th variable. This value must be checked against the values of the $k - 1$ past variables until we find an incompatible value pair, or until all $k - 1$ values have been checked. Since each consistency check has a probability $p$ of success, the average number of consistency checks at each node at level $k$ is

$$(k - 1)p^{k-1} + \sum_{i=1}^{k-1} i p^{i-1} (1 - p)$$

The probability that all $k - 1$ consistency checks subsequently succeed is $p^{k-1}$; hence, its contribution to the average number of is $(k - 1)p^{k-1}$. If the last check does not succeed its contribution is $(k - 1)p^{k-2}(1 - p)$. Analogous reasoning holds for the last but one, etcetera, leading to the summation in equation (6). Expression (6) equals $\frac{1 - p^{k-1}}{1 - p}$.

For the average number of consistency checks at the $k$-th level we multiply the average number of nodes in the search tree at this level by the average number of consistency checks at each node:

$$M^k p^{(k-1)(k-2)/2} \frac{1 - p^{k-1}}{1 - p}$$

The average total number of consistency checks $C_{tot}$ is the summation of the average number of consistency checks for each level $k$, for $k$ ranging from 1 to $N$.

$$C_{tot} = \sum_{k=1}^{N} M^k p^{(k-1)(k-2)/2} \frac{1 - p^{k-1}}{1 - p}$$

Next to the average total number of consistency checks we also can derive a prediction for the number of solutions of a CSP problem from the model by Haralick and Elliott. Equation (5) states that at level $k$ in the search tree for a CSP problem with $N$ variables and $M$ values, the average number of consistent value assignments (partial solutions) is $M^k p^{(k-1)/2}$. All partial solutions at level $N$ in the search tree represent full solutions, hence the average number of solutions $= M^N p^{N(N-1)/2}$

This formula is interesting, since it may be used to predict the location of the phase transition for a CSP problem: both Smith [14] and Williams and Hogg [19] assert that the point where the average number of solutions equals one, i.e., $M^N p^{N(N-1)/2} = 1$ can be used as a definition of the phase transition. From this equation we can derive that the probability $p_{trans}$ at which the phase transition will occur is

$$p_{trans} = M^{1/N^{N-1}}$$

$$M^k p^{(k-1)/2}$$
2.4 Difference in costs between first solution and average solution

We note that the computation of consistency checks above is based on an estimation of the size of the full search tree. If we are interested in the first solution only, the exploration of a part of this tree might be sufficient. Of course, one could consider $C_{tot}$ divided by the average number of solutions to be an estimate for the cost of a first solution. This is essentially the approach of Williams and Hogg [19, 20, 21]. However, the estimate is not accurate for all values of $p$, e.g., it is not well defined if the average number of solutions is less than one. Moreover, there are cases where the average solution cost differs considerably from the cost of the first solution. We show such a difference by a small example. Assume a CSP problem with 3 variables, with 2 values each, and $p = 1$ (i.e., each consistency check between a pair of values from two different variables succeeds). In figure 1 we have presented the problem as a search graph, where each node represents a different value assignment. If we use standard backtracking to search through this tree to find all solutions, we must carry out consistency checks every time we arrive at a new node. These consistency checks will be made against the values of all past variables. In figure 1, the number at each node indicates how many consistency checks will be carried out at that node.

![Figure 1: An example search tree, $N = 3$, $M = 2$, $p = 1$.](image)

In total, 20 consistency checks will be made to find all 8 solutions (since $p = 1$, all leaf nodes represent solutions.) This is an average of $5/2$ consistency check per solution. From figure 1 we see that the actual costs of a first solution amount to 3 consistency checks. It is obvious why this difference occurs: when averaging costs, all consistency checks made at an internal node in the tree are distributed over all solutions in the subtree of which this node is the root. Whereas, when searching for a first solution, all costs made at internal nodes contribute to the costs of the first solution. Therefore, at least for $p$ approaching 1, the cost of finding a first solution will be higher than the average cost per solution.

Another illustration of this difference directly emerges from the model by Haralick and Elliott. From equation 6 with $p = 1$, i.e., the average number of consistency checks at each node at level $k$ is $(k - 1)$, we can arrive at a version of equation 8 for $C_{tot}$ with $p = 1$. Dividing $C_{tot}$ by $M^N$ (the number of solutions for $p = 1$) we find that the average cost of a
solution tends to
\[
\frac{M^2(N-1) - MN + M^{(2-N)}}{(M-1)^2}
\text{as } p \text{ tends to 1}
\] (11)

Analyzing the standard-backtracking algorithm suggests however that for \( p = 1 \) a first solution will be found after \( N(N - 1)/2 \) consistency checks. We therefore conclude, that the estimate of formula (11) is not suitable as an accurate prediction of the average number of consistency checks for finding a first solution.
3 A new model for standard backtracking, first solution

The difference between the costs of finding a first solution and the costs of an average solution justifies a new approach to the prediction of costs of a first solution. Below, we present in two subsections a model for the average costs of finding a first solution when using a standard-backtracking algorithm. In 3.1, we examine how the probability of a successful value assignment relates to the probability of a single successful consistency check. Then, we investigate how the probability of a successful search from any level in the search tree relates to a successful value assignment. In 3.2, we calculate the average costs of the relations above. The set of equations is complex, and does not lead to a closed-form solution. However, by implementing these equations in a computer program (which is provided as an on-line appendix to this article), we arrive at the desired predictions. The reader is invited to use the on-line appendix in order to repeat our research.

3.1 Success and failure probability when searching

When considering the backtracking search process in detail, we distinguish between immediate and eventual success as well as between immediate and eventual failure. The latter distinction is related to the way a value assignment can fail:

1. It can fail immediately; if a value assigned to the $k$-th current variable is incompatible with any of the current values of the $k - 1$ past variables, the standard backtracking algorithm immediately rejects it.

2. It can fail eventually; if a value assigned to the $k$-th current variable is compatible with the current values of all past variables, the standard-backtracking algorithm provisionally accepts it; but if further search is unable to find a full solution with this set of $k$ variables and their current values, the value will eventually be retracted.

For the development of our model, we introduce the following definitions:

$p_{v,if}(k, j)$ is the probability that the assignment of the $j$-th value to the $k$-th current variable is an immediate failure ($if$).

$p_{v,is}(k, j)$ is the probability that the assignment of the $j$-th value to the $k$-th current variable is an immediate success ($is$), i.e., is not an immediate failure.

$p_{v,ef}(k, j)$ is the probability that the assignment of the $j$-th value to the $k$-th current variable is an eventual failure ($ef$), i.e., the value assigned will be retracted later on in the search process.

$p_{v,es}(k, j)$ is the probability that the assignment of the $j$-th value to the $k$-th current variable is an eventual success ($es$), i.e., the value assigned will not be retracted later on in the search process.

The subscript $v$ indicates that these probabilities are associated with single value assignments. Arguments $k$ and $j$ range over all variables and values, respectively ($1 \leq k \leq N$, $1 \leq j \leq M$).
A backtracking algorithm attempts all possible values of the current variable before it backtracks to a past variable. The sequence of attempted value assignments resides on the same level in the search tree. This is illustrated in figure 2. The assignment of the \( j \)-th value of the \( (k-1) \)-th variable can fail (F) immediately, or succeed (S) for the moment. If it succeeds, the value assignments of the \( k \)-th variable are tried. Any value assignment again may fail immediately or fail deeper in the search tree. If all \( M \) value assignments at level \( k \) fail, the algorithm backtracks to the next value assignment at level \( k-1 \). (Curved arrows show where the search process continues after backtracking.)

![Figure 2: General search behavior of CSP backtracking algorithms.](image)

The above gives rise to the introduction of the following probabilities with subscript \( l \) (for level):

- \( p_{i,ef}(k) \) is the probability that search from level \( k \) onwards will eventually fail, i.e., the search process must eventually backtrack to level \( k-1 \) or lower.
- \( p_{i,es}(k) \) is the probability that search from level \( k \) onwards will eventually succeed, i.e., a solution exists with the \( k-1 \) past variables having their current values.

For standard backtracking, we formulate the relations between the probabilities defined in this subsection in three steps. First, we present the relation between \( p_{i,iz}(k, j) \), \( p_{i,if}(k, j) \) and the probability \( p \) of a successful consistency check. Second, we show how the probability of eventual failure of a value assignment, \( p_{i,ef}(k, j) \), and the probability of its eventual success, \( p_{i,es}(k, j) \), are related to the immediate and level types of probabilities. And third, we recursively express the probabilities associated with search on a level, \( p_{i,ef}(k) \) and \( p_{i,es}(k) \), into probabilities associated with value assignments.

We start with the probabilities \( p_{i,iz}(k, j) \) and \( p_{i,if}(k, j) \). The values of \( p_{i,iz}(k, j) \) and \( p_{i,if}(k, j) \) depend on the search algorithm employed. In standard backtracking, a value assigned to the \( k \)-th current variable will be checked against the current values of all \( k-1 \) past variables until either an incompatible value pair has been found or all \( k-1 \) values have been checked. A value assignment at level \( k \) has immediate success if the \( k-1 \) consistency checks with the past variables all succeed. Since we assume that the probability of success of all these consistency checks is independent, this leads to equation (12). Conversely, a value
assignment immediately fails if any one of the \( k - 1 \) consistency checks fails. After a failing consistency check no subsequent check needs to be carried out. Analogously to equation (6), the reasoning above leads to equation (13).

\[
\begin{align*}
    p_{v, is}(k, j) & = p^{k-1} \\
    p_{v, if}(k, j) & = (1 - p)^{k-1} \\
    & = 1 - p_{v, is}(k, j)
\end{align*}
\] (13)

Equations (12) and (13) show that in our model the immediate success probability and the immediate failure probability are independent of \( j \), i.e., independent of the number of value assignments to the current variable. Henceforth, we will omit the argument \( j \) and use \( p_{v, is}(k) \) and \( p_{v, if}(k) \).

The second step in our presentation is to derive values for \( p_{v, es}(k, j) \) and \( p_{v, ef}(k, j) \). For \( k = 1, \ldots, N - 1 \), every value assignment at level \( k \) can fail in two independent ways: either the value assignment fails immediately, or it succeeds but the search fails (eventually) on the next level. At the deepest level \( N \), search can only fail if it fails immediately. Hence, for backtracking search strategies, the probability of eventual failure is governed by the following recursive equation:

\[
p_{v, ef}(k, j) = \begin{cases} 
    p_{v, if}(k) + p_{v, is}(k) p_{l, ef}(k+1) & \text{for } k = 1, \ldots, N - 1 \\
    p_{v, if}(N) & \text{for } k = N
\end{cases}
\] (14)

For a value assignment to be an eventual success, it must be compatible with the values of past variables (immediate success) and the subsequent search process deeper in the tree may not fail (i.e., no backtracking beyond this value assignment). Since these two events are independent in our model, the probability of eventual success for a value assignment at level \( k \) is

\[
p_{v, es}(k, j) = \begin{cases} 
    p_{v, is}(k) p_{l, es}(k+1) & \text{for } k = 1, \ldots, N - 1 \\
    p_{v, is}(N) & \text{for } k = N
\end{cases}
\] (15)

Again we remark that since the right-hand expressions of equations (14) and (15) are independent of \( j \), we omit \( j \) as an argument from now on. In the remainder of the article, we use \( p_{v, ef}(k) \) and \( p_{v, es}(k) \) instead of \( p_{v, ef}(k, j) \) and \( p_{v, es}(k, j) \), respectively.

The third step is to derive values for \( p_{l, ef}(k) \) and \( p_{l, es}(k) \). For the eventual success or failure of the search process at a level of the search tree we look again at figure 2. We consider first what happens at the last level of the search tree. At the last and deepest level of the search tree, search can only fail if all value assignments of the \( N \)-th variable fail (by then, the \( N - 1 \) past variables all have compatible values). This is represented in equation (16). Conversely, the search succeeds at level \( N \) if there is any value assignment at this level that succeeds. Equation (17) is derived analogously to equation (6). If we find a value for the \( N \)-th variable that is compatible with all the values of the \( N - 1 \) past variables, we have found a solution.

\[
p_{l, ef}(N) = [p_{v, if}(N)]^M
\] (16)
\[ p_{l,es}(N) = p_{v,es}(N) + p_{v,ef}(N)p_{v,es}(N) + \ldots + (p_{v,ef}(N))^{M-1}p_{v,es}(N) \]
\[ = p_{v,es}(N) \sum_{i=0}^{M-1} [p_{v,ef}(N)]^i \]
\[ = 1 - p_{l,ef}(N) \] (17)

The situation at all levels in the search tree is in fact similar to the situation at the deepest level, provided that we replace immediate success and failure by eventual success and failure.

Equation (19) expresses the notion that search can only fail at a level \(k\) in the search tree, if all \(M\) value assignments eventually fail.

\[ p_{l,ef}(k) = [p_{v,ef}(k)]^M \] (19)

For the search to succeed at any level \(k\), either the first value assignment is an eventual success (leads to a solution), or the first value assignment eventually fails but the second one succeeds, etc. This leads to equation (20). (In passing, we note that the equations (18) and (21) are used in subsection 4.2.)

\[ p_{l,es}(k) = p_{v,es}(k) + p_{v,ef}(k)p_{v,es}(k) + \ldots + (p_{v,ef}(k))^{M-1}p_{v,es}(k) \]
\[ = p_{v,es}(k) \sum_{i=0}^{M-1} [p_{v,ef}(k)]^i \]
\[ = 1 - p_{l,ef}(k) \] (20)

Finally, we note that the set of equations (16)-(21) recursively describes our failure probability \(p_{l,ef}(k)\) and success probability \(p_{l,es}(k)\).

The formulas presented above are similar to some formulas presented in [16], but harder to express in a closed form (if at all possible) because of parameter \(M\). Fortunately, it is possible to understand the general behavior by combining equations, e.g., using equations (12)-(21), and starting with equation (19) we straightforwardly arrive at the following

\[ p_{l,ef}(k) = \begin{cases} (1 - p^{k-1}(1 - p_{l,ef}(k + 1)))^M & \text{for } k = 1, \ldots, N - 1 \\ (1 - p^{N-1})^M & \text{for } k = N \end{cases} \] (22)

Obviously, the recursive set of equations in (22) has a limit point in \(p_{l,ef}(k) = 1\). The overall behavior of \(p_{l,ef}(k)\) can best be observed on a relatively small problem, e.g., on a CSP problem with \(N = M = 10\).

In figure 3 we see that for small values of \(p\) (in this case, for \(p < 0.6\)), the probability of failure is relatively small at the deepest level in the search tree. (There, the search can only fail locally, not deeper in the tree.) As we go backwards, from level 10 to 9, etcetera, the probability of failure increases. If this failure probability reaches a value of one at any level, the search as a whole must fail (e.g., see \(p = 0.5\)). If however the maximum is even slightly below one (e.g., see the case for \(p = 0.6\)), there is a probability that an overall solution can be found. The latter probability is expressed by \(1 - p_{l,ef}(1)\), so if \(p_{l,ef}(1) < 1\) in the figure, a solution to the CSP problem may be found. For larger values of \(p\) (in the example, \(p = 0.7\) and up), their maximum \(p_{l,ef}(k)\) value will be progressively lower, until eventually the failure probability will be decreasing as \(k\) decreases from 10 to 1 (see \(p = 0.85\)).
3.2 Average number of consistency checks

Below, we introduce the notion of events described by the probabilities of subsection 3.1. All events are associated with elementary steps in the backtracking search process. Each step has a certain cost, measured in terms of consistency checks. We define the following costs:

\( C_{v_{k}}(k) \), the average cost associated with a value assignment at level \( k \) in the search tree, given that the assignment fails immediately.

\( C_{v_{k}}(k) \), the average cost associated with a value assignment at level \( k \) in the search tree, given that the assignment succeeds for the moment, i.e., is not an immediate failure.

\( C_{v_{k}}(k) \), the average cost associated with a value assignment at level \( k \) in the search tree, given that the assignment eventually will be retracted.

\( C_{v_{k}}(k) \), the average cost associated with a value assignment at level \( k \) in the search tree, given that the assignment will not be retracted later on in the search process.

\( C_{l_{k}}(k) \), the average cost associated with searching from level \( k \) in the search tree onward, given that the search process eventually must backtrack to level \( k - 1 \).

\( C_{l_{k}}(k) \), the average cost associated with searching from level \( k \) in the search tree onward, given that a solution will be found without having to backtrack to level \( k - 1 \).

All these costs are conditional costs, i.e., costs associated with a given event, e.g., a value assignment has failed. For the purpose of our research, we are interested in \( C_{l}(k) \), the average cost of searching the tree for a first solution at level \( k \) and deeper. In terms of conditional
costs and probabilities associated with an event, the relation between \( C_l(k) \) and \( C_{lf}(k) \) and \( C_{l^2}(k) \) is given by

\[
C_l(k) = p_{i_e}(k)C_{l^2}(k) + p_{i_f}(k)C_{lf}(k)
\]

(23)

To calculate the costs of success (or failure) at any level in the search tree, we start with the cost of the basic step: a value assignment. The average cost of immediate success or failure of a value assignment for standard backtracking is determined by the events associated with equations (12) and (13), (see also equation (6)). A value assignment at level \( k \) can have immediate success only if the \( k-1 \) consistency checks with the current values of the past variables all succeed. Therefore, the cost of immediate success for a value assignment is proportional to \( k-1 \). According to our definition of costs, it can be measured as \( k-1 \) (see equation (24)).

\[
C_{vf}(k) = k - 1
\]

(24)

The average cost of an immediately failing assignment is determined by the weighted summation of all possible ways to fail, divided by the probability of failing. The division is necessary since \( C_{vf}(k) \) is a conditional cost. The result is shown in equation (25).

\[
C_{vf}(k) = \frac{\sum_{j=1}^{k-1} j p^{j-1} (1 - p)}{p_{v:lf}(k)} = \frac{1}{1 - p} - \frac{(k - 1) p^{k-1}}{1 - p^{k-1}}
\]

(25)

The average cost of a value assignment given that it eventually fails, \( C_{vf}(k) \), is a weighted sum of the costs of the two different ways a value assignment can fail, viz. \( if \) and \( ef \) (cf. the events associated with equation (14)). Each cost is weighted by the relative probability of its occurrence. At level \( N \) there is only one way to fail.

\[
C_{vf}(k) = \begin{cases} 
  p_{v:if}(k)C_{vf}(k) + p_{v:is}(k)p_{i:ef}(k + 1) [C_{vf}(k) + C_{lf}(k + 1)] & \text{for } k = 1, \ldots, N-1 \\
  C_{vf}(N) & \text{for } k = N
\end{cases}
\]

(26)

The average cost of a value assignment given that it eventually succeeds, \( C_{ve}(k) \), is the sum of the direct cost of a successful value assignment and the costs of a successful search deeper in the tree (cf. equation (15)).

\[
C_{ve}(k) = \begin{cases} 
  C_{ve}(k) + C_{vf}(k + 1) & \text{for } k = 1, \ldots, N - 1 \\
  C_{ve}(N) & \text{for } k = N
\end{cases}
\]

(27)

Now we can express \( C_{lf}(k) \) and \( C_{le}(k) \) necessary for equation (23). For the search to fail at level \( k \), all \( M \) value assignments must fail (cf. equation (19)). Therefore the average cost of a failing search at level \( k \), given that it fails, is

\[
C_{lf}(k) = MC_{vf}(k)
\]

(28)
For the average costs of an eventual success at level $k$ we consider the different ways a first solution may be found (see equation (20)). If the first value assignment at level $k$ eventually is a success, we have only an average cost of $C_{v|e|s}(k)$. If the first one fails in the end but the second one succeeds, we have incurred on average the failure cost $C_{v|f|f}(k)$ for the first value assignment, and the cost $C_{v|f|s}(k)$ for the successful second assignment. The costs of each way in which a solution can be found must be weighted by the probability of its occurrence.

$$p_{v|e|s}(k)C_{v|e|s}(k)\ = \ p_{v|e|s}(k)C_{v|f|f}(k) + p_{v|e|f}(k)p_{v|e|s}(k)\left[C_{v|f|f}(k) + C_{v|e|s}(k)\right] + \ldots + p_{v|e|f}(k)^{M-1}p_{v|e|s}(k)\left[(M-1)C_{v|f|f}(k) + C_{v|e|s}(k)\right]$$

(29)

$$= \ p_{v|e|s}(k)C_{v|e|s}(k)\sum_{i=0}^{M-1} [p_{v|e|f}(k)]^i + p_{v|e|s}(k)C_{v|f|f}(k)\sum_{j=0}^{M-1} j[p_{v|e|f}(k)]^j$$

(30)

Combined with the probability equations (12) to (21) from subsection 3.1, we now have a set of equations that describe the average cost of searching a tree under the assumptions of our model (cf. subsection 2.2).

### 3.3 A computer program

Rather than trying to solve the recurrent relations or to find an approximate analytical solution, we have encoded them in a computer program, available to the readers as an on-line appendix. This program may be used to predict search behavior on a given CSP representation of a problem, provided we know $N$, $M$, and $p$. For the determination of the average cost of finding a first solution, using standard backtracking, we compute the average cost of searching from the first level in the tree, $C_{v}(1)$. The average cost given that a solution is found is represented by $C_{v|e|s}(1)$, and the probability that a solution will be found by $p_{v|e|s}(1)$. By computing this probability over a range of values for $p$ for a given problem set, we may be able to determine the phase transition. By definition, the crossover point occurs where the probability of finding any solution is 1/2. Some experimental results are given in section 6.
4 A new model for min forward checking, first solution

In subsection 2.1 we have described the algorithm for min forward checking. In this section we provide a model for the average cost of finding a first solution for CSP problems when using the min-forward-checking algorithm. To accommodate the algorithm we adapt the model for standard backtracking in two ways. First, the costs and probabilities associated with a single value assignment are adapted. Henceforth, these are called the immediate parameters. Second, the level equations are adapted. As with the model for standard backtracking, our purpose is to present a simple model of a CSP problem representation for predictions. Again, the model is encoded in a C program and available as an on-line appendix.

4.1 Adaptation of the immediate parameters

To create a model for finding the first solution of a CSP problem using min forward checking, we must adapt the immediate parameters from the model for standard backtracking. This adaptation is necessary since the two algorithms differ in their basic value assignment step. For standard backtracking, equations (12)–(13) and (24)–(25) define the immediate parameters. For min forward checking we need some additional definitions before we can define the immediate parameters. We first consider the additional definitions for forward checking, then for min forward checking.

As described in subsection 2.1, in the forward-checking algorithm each value assigned to the current variable will be checked against all values of future variables for compatibility. Here we define a check sequence as the basic step in forward checking: a value (of the current variable) is checked against all remaining values of a (future) variable. When forward checking has successfully assigned a value to the \(i\)-th variable, the remaining values of the future variables are compatible with the current values of the \(i-1\) past variables and the current variable. Each remaining value has therefore "survived" \(i\) consistency checks. We first introduce

\[
q(i, j) \text{ as the probability that a future variable has } j \text{ remaining possible values, after testing values for the } i\text{-th current variable, i.e., the probability that exactly } j \text{ values survive after } i \text{ check sequences.}
\]

Since we assume that the probability of success of a consistency check is independent of the two variables involved, and all variables initially have \(M\) possible values, the probability \(q(i, j)\) is independent of the future variable involved. At the start, all variables have \(M\) values:

\[
q(0, j) = \begin{cases} 0 & \text{for } j = 0, \ldots, M - 1 \\ 1 & \text{for } j = M \end{cases}
\]  \(31\)

When the \(i\)-th variable has a current value that is acceptable after forward checking, we know that all future variables have at least one possible value. We therefore introduce

\[
q'(i, j) \text{ as the probability that a future variable has } j \text{ remaining possible values, after testing values for the } i\text{-th current variable, and under the condition that we can continue with}
\]
the $(i+1)$-th variable. So it is the probability that a future variable has $j$ values left after $i$ variables have been assigned compatible values.

This probability is a normalized version of probability $q(i, j)$:

$$q'(i, j) = \frac{q(i, j)}{1 - q(i, 0)} \quad \text{for } j = 1, \ldots, M$$

(32)

since $1 - q(i, 0)$ is the probability that a variable has at least one value remaining after $i$ check sequences.

We can now recursively describe $q(i, j)$:

$$q(i, j) = \sum_{m=\max(j,1)}^{M} q(i - 1, m) \binom{m}{j} p^j (1 - p)^{m-j} \quad \text{for } j = 0, \ldots, M$$

(33)

This equation can be interpreted in the following way. Assume that a future variable has $m$ values left, after successful value assignments to $i - 1$ variables. When assigning a value to the $i$-th variable, these $m$ values are all subjected to a consistency check, and the number of ways that $j$ of these checks succeed equals $\binom{m}{j}$. Since in our model any consistency check will succeed with probability $p$, each of those ways has a probability $p^j (1 - p)^{m-j}$ of occurring. To evaluate $q(i, j)$ correctly, we must weight the occurrence of $m$ values after $i - 1$ check sequences by its probability $q'(i - 1, m)$.

We now continue with min forward checking. When min forward checking has successfully assigned a value to $i - 1$ variables, a future variable with the least remaining values will be selected as the next current variable. The values remaining for this variable are all candidates for the next value assignment. To estimate how many value assignments are possible for this new current variable (before backtracking to the $(i-1)$-th variable), we introduce

$q_{\text{min}}(i, j)$ as the probability that the future variable with the least remaining possible values has exactly $j$ possible values, after $i$ variables have been assigned compatible values.

To simplify the mathematical expression for $q_{\text{min}}(i, j)$, we also introduce

$Q(i, j)$ as the probability that a future variable has more than $j$ remaining possible values, after $i$ variables have been assigned compatible values.

Based on the definition of $q'(i, j)$, a value for $Q(i, j)$ can be easily computed from

$$Q(i, j) = \sum_{m=j+1}^{M} q'(i, m)$$

(34)

After $i$ variables have been successfully assigned a value, $N - i$ future variables remain. For $j$ to be the minimum number of values among the $N - i$ future variables, there must be at least $h$ ($h > 0$) of these future variables with exactly $j$ values, and all others must have more than $j$ values. From the definitions given, it follows that

$$q_{\text{min}}(i, j) = \sum_{h=1}^{N-i} \binom{N-i}{h} [q'(i, j)]^h [Q(i, j)]^{(N-i) - h}$$

(35)
where $j$ ranges from $1, \ldots, M$. Having introduced the additional definitions, we remark that we deliberately ignored any subtle dependencies that might arise by the method of selection of the current variable. We once more stress that our assumption still is that the probability of success of any consistency check is independent of variables and values involved, and independent of past processing.

The additional definitions enable us to define the immediate parameters for min forward checking. We start with $p_{v,i}(k)$. For a value assignment to the $k$-th variable to succeed, all $(N-k)$ future variables must have at least one value. Since $1 - q(i, 0)$ is the probability that a future variable has one or more values remaining after $i$ check sequences,

$$p_{v,i}(k) = \prod_{j=k+1}^{N} \left[ 1 - q(k, 0) \right] = \left[ 1 - q(k, 0) \right]^{N-k}$$  \hspace{1cm} (36)

Equation (13), being $p_{v,i}(k) = 1 - p_{v,i}(k)$, still determines the immediate failure probability.

To adapt the equations (24) and (25) for the immediate costs, we here introduce an extra cost parameter for convenience:

$C_v(i)$ is the average cost of a consistency check sequence (pertaining to a single future variable), after $i - 1$ variables have successfully been assigned values.

The average cost pertaining to a future variable that fails (i.e., has no possible values left after the current value assignment) is

$$C_v(i) = \sum_{m=1}^{M} mq'(i-1, m)$$  \hspace{1cm} (37)

since all $m$ possible values of this variable must be checked to be sure that none is compatible. However, equation (37) also gives the average cost for a future variable that has at least one compatible value. The reason for this is that forward checking continues to check all its values for later assignments, even after it has found a compatible value. The result is that, after the checks, all remaining values of this future variable are compatible with the current value assignment. (This means that if a future variable is selected as the next current variable, any of its remaining values may be selected as a current value, since they are all compatible with the values of past variables.)

For min forward checking, we therefore replace equation (24) by

$$C_{v,k}(k) = (N-k)C_v(k) = (N-k) \sum_{m=1}^{M} mq'(k-1, m)$$  \hspace{1cm} (38)

i.e., in case of a successful value assignment to the $k$-th variable all $(N-k)$ future variables have all their values checked.

Moreover, we replace equation (25) for the average cost of an immediately failing value assignment by

$$C_{v,f}(k) = \frac{C_v(k)q(k, 0) \sum_{h=1}^{N-k} h \left[ 1 - q(k, 0) \right]^{k-1}}{p_{v,f}(k)}$$  \hspace{1cm} (39)

since we only check future variables until we find one variable that has no values left.
4.2 Adaptation of the level equations

For min forward checking we also must adapt the level equations (16)–(21) and (28)–(30), since we may no longer assume that each variable has the initial $M$ possible values left when it becomes the current variable. The probability distribution $q_{\min}(k, m)$ determines the number $m$ of remaining values of the $k$-th current variable. Using this probability distribution we adapt equation (19) for $p_{i,ef}(k)$ to

$$p_{i,ef}(k) = \sum_{m=1}^{M} q_{\min}(k-1, m)[p_{v,ef}(k)]^m$$

(40)

i.e., search failure on a level occurs only when all value assignments fail, but each occurrence of the number of values must be weighted by its probability. For $p_{i,es}(k)$, we can still use equation (21): $p_{i,es}(k) = 1 - p_{i,ef}(k)$. Consequently, we adapt equation (28) to

$$C_{l}\|_{ef}(k) = C_{v}\|_{ef}(k) \sum_{m=1}^{M} mq_{\min}(k-1, m)$$

(41)

Again, each occurrence of $m$ is weighted by its probability. Equation (30) is therefore adapted in a similar fashion to

$$p_{i,es}(k)C_{l}\|_{es}(k) = \sum_{m=1}^{M} q_{\min}(k-1, m) \left[ p_{v,es}(k)C_{v}\|_{es}(k) \sum_{i=0}^{m-1} (p_{v,ef}(k))^i + p_{v,es}(k)C_{v}\|_{ef}(k) \sum_{j=0}^{m-1} j(p_{v,ef}(k))^j \right]$$

(42)

We remark that a model for (regular) forward checking can be obtained by replacing the parameter $q_{\min}(k-1, m)$ in equations (40), (41), and (42) by $q'(k-1, m)$. This replacement is possible, since forward checking selects its next current variable lexicographically (cf. subsection 2.1).

4.3 A computer program

The new model, i.e., with the adaptations described above, provides us with an analytical tool for predicting the average costs of searching with min forward checking. We have implemented the model in a computer program, which is available as an on-line appendix to the reader. As in the model for standard backtracking, $C_l(1)$ gives the average cost of solving a CSP problem, since the search process starts at level 1 in the search tree. $C_{l}\|_{es}(1)$ supplies the average cost given that a solution is found, and $p_i,es(1)$ the probability that a solution will be found. Again, a particular area of interest is around $p_{i,es}(1) = 0.5$, where a crossover occurs between a region where most problem instances have solutions ($p_{i,es}(1) > 0.5$), and a region where most problem instances are insoluble. Experimental results are given in section 6.
5 Model predictions

In this section we describe the behavior of various model predictions. First, we relate our model for standard backtracking to the model by Haralick and Elliott [6]. Then, we demonstrate that our models are in competition. Dependent on the value of $p$ a search algorithm may be selected that is efficient for a given CSP representation.

5.1 Modelling standard backtracking: first versus all solutions

When comparing our model for standard backtracking to the model by Haralick and Elliott, we must bear in mind that the latter one predicts the costs for all solutions, whereas our model does so for the first solution only (cf. section 3). Figure 4 illustrates the relation between the two models for standard backtracking with $N = M = 10$. The dependence of the average costs on parameter $p$ is depicted for three cases (all solutions, a first solution, given a solution is found).

![Figure 4: Relation between models for search behavior of standard backtracking.](image)

The top line in figure 4 represents $C_{tot}$, the average cost of finding all solutions, according to the model by Haralick and Elliott. The average cost of finding a first solution as predicted by $C_1(1)$ coincides with this line up to point $A$, thereafter it slopes downward. The line starting at point $B$ represents the average cost given that a solution is found (as predicted by $C_{es}(1)$). The latter two lines coincide from point $C$ onward. From there, the probability of finding a solution virtually equals one (see figure 7).

Obviously, for low values of $p$, it is unlikely that any solution will be found (cf. equation (9)). If the average number of solutions is much less than one (say, 0.01) it is expected that out of a sample of 100 problem instances only one instance will have a solution. This means that the full search tree will be examined in almost all sample cases. Thus, the cost of searching
for all solutions or for the first solution only hardly differ. This explains why \( C_{\text{tot}} \) and \( C_l(1) \) coincide up to a certain extent \( (p \approx 0.55) \).

As \( p \) increases, the average number of solutions also increases. After point A, the two lines representing the costs of searching gradually diverge, since the search process for a first solution stops the consistency checking after a solution has been found. The average cost for finding a first solution is claimed to attain a maximum at the value of \( p \) for which the average number of solutions in the tree exactly equals one. (This claim of coincidence is as yet unsupported; experiments have shown that the maximum is at least very near. Similar claims have been made by Williams and Hogg [19, 21] and Smith [15].)

As stated above, the average cost of finding a first solution \textit{given} that a solution is found is also presented in figure 4. Since this cost is irrelevant if it is very unlikely to find a solution, we started the line, rather arbitrarily, in point B, where the probability of finding a solution \( (p_{\text{fis}}(1)) \) first exceeds \( 1 \times 10^{-2} \). (Smith defines the region where the probability of finding a solution is between 0.01 and 0.99 as the \textit{mushy region} [15].) The average cost of finding a solution and the average cost of finding a solution \textit{given} that a solution is found coincide from the point where the probability of finding a solution has become one (in figure 4, point C).

### 5.2 Model predictions for the average costs

In figure 5 we show model predictions for the average cost of finding a first solution using standard backtracking and min forward checking. For CSP problems with \( N = M = 10 \), the solid line shows the predicted average cost for searching for a first solution using standard backtracking. The cost is given as a function of the probability \( p \) of a successful consistency check. The dashed line shows the predicted average cost using min forward checking.

![Figure 5: A comparison of model predictions for different algorithms.](image-url)
We see in figure 5 a range of values for $p$ where using min forward checking is more efficient than using standard backtracking. However, for $p$ close to 1 standard backtracking is more favorable. (We made these observations also for various other values of $N$ and $M$ not reported here.) The latter result can be readily explained: based on the description of the algorithms, we conclude that as $p$ tends to 1 (almost any value assignment succeeds immediately), the number of consistency checks to find a first solution approaches $N(N - 1)/2$ for standard backtracking. For min forward checking, however, this number approximates $MN(N - 1)/2$, hence a factor $M$ larger. (This is because we check the value of the current variable against all values of future variables.)

![Figure 6: Model predictions for the average cost of backtracking for various problem sizes.](image)

In figure 6 we show the behavior of the model for standard backtracking for various sizes of $N$ and $M$. The lines show the predicted average cost for finding a first solution. The predictions are for CSP problems with $N$ variables and $M$ values, with $N = M = 10, 20, 30, \text{ and } 40$, respectively (the larger the problem size, the higher the costs). Vertical lines indicate at what values of $p$ the peak in search cost is predicted to occur. They are based on two different definitions. First, Smith [15] and Williams and Hogg [19, 21] conjecture that the maximum search cost will occur when the expected number of solutions equals one. For our model, the corresponding probability $p_{\text{trans}}$ can be calculated from equation 10. Second, for satisfiability problems, Mitchell, Selman and Levesque [10] observed that the maximum in search effort coincides with the point where half of the randomly generated problem instances have a solution. This point is called the crossover point by Crawford and Auton [2], and can be determined in our model by computing the value of $p$ where $p_{l,ex}(1) = 0.5$. For the two smaller problems ($N = 10, M = 10$ and $N = 20, M = 20$) the vertical lines for $p_{\text{trans}}$ and the crossover point coincide (see figure 6). In the cases with $N = 30, M = 30$ and $N = 40, M = 40$ the left vertical lines on each curve represent $p_{\text{trans}}$, the right ones the crossover points.

From the crossover points we see that as the problem size increases, the probability of success for a consistency check necessary to find any solution increases. Near $p = 1$ there is a nearly
level section in every curve. As $N$ and $M$ increase, this section becomes smaller (see again figure 6). In this level region, the probability of success of any value assignment is high enough for the search process to find a solution rather fast, without much backtracking. If the value of $p$ is lower, the cumulative effect of failing value assignments will cause many subtrees to be examined before a solution is found. (If the probability $p$ is very low, the cost is again low, because the search process quickly detects that a subtree does not lead to a solution.)

As the problem size increases, the peak in average search costs for a first solution becomes sharper. This was already observed experimentally by Cheeseman, Kanefsky and Taylor [1]. Moreover, figure 6 shows that for the larger problems the predicted location of the peak at $p_{\text{trans}}$ coincides with the actual peak in the full model. However, it is very surprising to see that although for the two smaller problems $p_{\text{trans}}$ and the crossover point agree, for the larger ones these points differ! It turns out that our model predicts that the peak in search costs will occur in a region where the probability of a solution is smaller than 0.5.

Considering figure 6 with a fixed probability $p$, e.g., $p = 0.83$, we see the following three phases in the search processes as the problem size increases:

1. For small problem sizes ($N, M < 20$), the cost of finding a first solution is in the nearly level section. As problem size increases, the cost increase is slightly more than quadratically.

2. For medium problem sizes ($N, M$ between 20 and 40), costs will rise exponentially, but a solution will still be found.

3. For large problem sizes ($N, M > 40$; not in figure 6), the probability that a solution will be found vanishes.

Once a CSP representation of a problem is given, the parameters $N$ and $M$ are known and the value of $p$ can be determined. A graph such as the one in figure 5 can then be used to decide which search algorithm is most efficient for the problem under investigation. A graph such as in figure 6 (or in figure 7) may help us to see whether a solution can be expected.
6 Experimental

In this section we compare the predictions for the average costs and for the probability of finding a solution to experimental results on random CSP problems, and on instances of the \(N\)-queens problem. Moreover, we show that the model for standard backtracking can be used to predict the phase transition for graph coloring problems.

6.1 Predicting the behavior of random CSP problems

Below we compare the predictions of our models with otherwise obtained measurements on random CSP problems. For given values of \(N = 10\) and \(M = 10\), we generated samples of 100 random CSP problems for selected values of \(p\). The problems were generated with the number of randomly placed binary constraint pairs, \(B\), corresponding exactly to the number required for the desired probability \(p\). This number \(B\) can be computed exactly using equation (2). The method of generating random CSP problems is equivalent to what Smith [14] calls “Model B”, and its main advantage is that any problem generated in this way has exactly the desired value of \(p\). Any variation in the results must therefore be attributed to other factors (the search method or the particular constraint graph).

![Figure 7: Model predictions and measured values for the probability of finding any solution.](image)

The problems with \(N = 10\), \(M = 10\) can be considered small, and therefore the phase transition is not as sharp as for larger problems. As it turns out, the size chosen illustrates better the limitations of the models than larger experiments. Moreover, larger experiments could not be carried out in sufficient quantities due to the inefficiency of our Lisp programs: at about 1300 consistency checks per minute on a Sun Sparc 5 for the more difficult problems, experiments would take very long to complete. (Many of the experiments by Prosser [13] have a comparable problem size: \(N = 8\), \(M = 10\).) As will be shown below, the sample size
of 100 is sufficient to illustrate the behavior of the average search cost with a 90% confidence interval. The results are shown in the figures 7 to 10.

First, we used our model for standard backtracking to predict the probability that a solution is to be found. In figure 7, the solid line shows this probability as a function of $p$, predicted by $p_{lcs}(1)$. For samples of 100 random CSP problems with $N = M = 10$ and $p$ fixed, the dots denote the fraction of the sample that actually had a solution. As figure 7 shows, measurements and predictions conform reasonably well. It appears that for random CSP problems $p_{lcs}(1)$ is an adequate estimate of the fraction of sample problems with a solution.

Second, for the costs of searching, the correspondence between model and measurements is illustrated in figure 8. The average cost of searching for a first solution (or showing that there is none) computed from the samples is denoted by a solid line. The dashed lines above and below it indicate the 90% confidence limits between which the actual average will lie. According to standard statistics texts, e.g., [22], we can say that with a probability of 90% the actual average will be between these bounds. The dotted line indicates the model predicted values for the average cost. Figure 8 also contains the minimum and maximum cost of finding a solution (or showing that there is none) as observed in the samples.

![Figure 8: Predicted and measured average cost of finding a first solution using backtracking.](image)

Haralick and Elliott [6] already showed that their model for finding all solutions conformed well to measurements on random CSP problems. Not surprisingly therefore, we see in figure 8 a close correspondence between the measurements and predictions in the region where our model coincides with the model by Haralick and Elliott (i.e., for $p \lesssim 0.55$ cf. figure 4). Although the shape of the curve in figure 8 is predicted accurately, the model-predicted values differ from the measured average in the region $0.6 \leq p \leq 0.85$. Notice that the size of the samples is sufficient to make this conclusion, with 90% confidence. We conjecture that the discrepancy between model and measurements is in the region where (1) a chance of finding a solution exists, and (2) the variance in results is very high. One could argue that in this region our assumptions (i.e., independence of past processing, independence of variables...
and values involved) break down. However, we believe it more likely that the mean-field approximation in our model (the fact that we use the computed average cost of level $k+1$ to compute the average cost at level $k$) is responsible for the discrepancy, since the variation is so high. For $p > 0.85$, the model predictions and the experimental values converge. If we had based our predictions for this region on the model by Haralick and Elliott no convergence would appear. Here we reiterate that there is a difference in costs between a first solution and average solution (cf. subsection 2.4).

Figure 9: Our model vs. the “strawman” prediction for the cost of the first solution, for $N = 10, M = 10$.

In figure 9, we have illustrated our prediction for the cost of a first solution, and the “strawman” prediction based on Haralick and Elliott’s prediction for the cost of all solutions (equation 8) divided by the expected number of solutions (equation 9). We see that as $p$ grows larger, the strawman prediction fails to predict accurately the costs, as already discussed in subsection 2.4. Since it is our aim to predict the costs for the full range of values for parameter $p$, this approach is not sufficient.

Moreover, we remark that the whole shape of the curve in figure 8 nicely corresponds to the curve based on data of random CSP problems as observed by Gaschnig (see figure 4.4.3-1 in [4]). Gaschnig saw from his observations that there is a peak in the average cost of finding a first solution around a degree of constraint $L = 0.6$. Our model predicts such a peak.

Third, figure 10 shows the results for the same random CSP problems, now solved by using min forward checking. The same conventions as in figure 8 hold. We note that the variation

\[ p = \frac{N}{N + 1} \times L. \]
in the costs is smaller, and the predictions are closer to the actual measured costs for values of $p \gtrsim 0.75$. Again, the region of least agreement is the region where a chance of finding a solution exists and the variation in search costs is high.

### 6.2 Predicting the behavior of the $N$-queens problem

In the previous subsection we have shown that our models predict various kinds of behavior for search processes on random CSP problems. Below, we investigate whether the models can also be applied to other classes of problems. As an example, we take the $N$-queens problem.

A condition for making predictions on the $N$-queens problem is knowing the values of $N$, $M$, and $p$ to be used in our models. We employed a CSP representation with each variable representing a row on the chess-board on which a queen must be placed. The values of this variable correspond to board positions in this row (lexicographically ordered). The number of variables $N$ then equals the number of rows on the chess-board, the number of values, $M$, is the number of files (columns). In [18], we derived a formula to compute the probability $p$ for this representation of the $N$-queens problem:

$$p = \frac{3N^2 - 7N + 2}{3N^2}$$

This formula is equivalent to the one mentioned in [14]. Alternatively, the value of $p$ could be measured directly from the CSP representation using equation (2).

To evaluate the model, experiments were performed for different board sizes $N$. For each board size $N$, a sample of 100 problem instances was solved using both standard backtracking...
and min forward checking. In each problem instance, the order of the variables and the order of values belonging to a variable was randomly permuted from the original problem. In this way, the influence of any particular ordering was diminished.

Figure 11: $N$-queens problem, predicted and measured cost of finding a first solution using backtracking.

The results on the average search effort using standard backtracking are shown in figure 11. As in figure 8, a solid line indicates the measured average cost of searching for a first solution. Above and below that line, dashed lines delimit the 90% confidence interval in which the actual average will lie. A dotted line indicates the model predicted values for the cost of finding a first solution. Also, the minimum and maximum costs observed in the samples are included in the figure. It is apparent that there is not a good agreement between theory and practice. Moreover, the 90% confidence interval shows that it is unlikely that more experiments will remedy this discrepancy. As for the reason of this discrepancy, we tested various hypotheses.

First, we made additional measurements (not shown here) to verify that the discrepancy is not caused by a dependence on the particular order in which the variables were searched. This was done by randomly rearranging the order of future variables whenever a new current variable was to be selected. Similarly, the order of values was dynamically rearranged on every new value assignment. The average cost measured in these experiments was the same as in the experiments with the given fixed evaluation order.

Second, we generated random CSP problems with the same number of variables, values, and constraint pairs as the $N$-queens problems of size $N$. We call these random CSP problems probabilistically equivalent (under parameter $p$) to the corresponding $N$-queens problems, since they have the same value for $p$. Measurements on these random CSP problems do agree with model predictions (see [18]), as could be expected from subsection 6.1. This leads us to conclude that there is more to the $N$-queens problem than just the number of constraints. This “more” can be termed problem structure. Smith [14] has suggested that the regularity of the constraint graph has a direct influence on the effort of finding a first
solution or showing that there is none. This leaves room for further research, on how to qualify this regularity and incorporate it into a model for predicting the search effort.

![Figure 12: N-queens problem, predicted and measured cost of finding a first solution using min forward checking.](image)

The results for min forward checking applied to these N-queens problems are illustrated in figure 12. The same conventions as in figure 11 hold. We see an excellent agreement between model predictions and observed values. This agreement may however be partly the result of the solution cost distribution. Notice that the difference between average solution cost and minimum solution cost is much smaller than for standard backtracking: min forward checking, by its dynamic selection of the next current variable, apparently tends to even out the solution costs.

### 6.3 Predicting phase transitions for graph coloring problems

For many NP-complete problems, there is a phase transition as an order parameter is varied. Cheeseman et al. [1] made the observation, that there is a transition from a region where many of the problem instances are easily soluble to a region where many of the problem instances are easily proved insoluble. They observed that the problems which are, on the average, hardest to solve occur between these two regions. Also, as problems become larger, the phase transition becomes sharper. It is therefore interesting to predict where the phase transition will lie for a given problem representation, since it will identify where the hard problems are. When considering the case of algorithm selection (as we do), however, the entire range of the order parameter is interesting.

In recent AI literature, many problems with a phase transition are discussed. We mention two in particular: k-satisfiability problems (see e.g., [1, 2]) and graph coloring problems [19]. Although the representation of a k-satisfiability problem as a CSP problem with binary constraints seems straightforward, in particular if $k = 2$, we have not been able to find a
representation in which the assumption underlying our model for \( p \) is sufficiently valid. In other words, our CSP representations of 2-SAT \( (k = 2) \) and 3-SAT \( (k = 3) \) violated the assumption that the probability of success of a consistency check should be independent of past processing, in such a way that the results were not accurate enough.

In the remainder of this subsection we shall show that our theory can be used to predict the phase transition point for graph coloring problems. Garey and Johnson [3] define the graph \( K \)-colorability problem as follows:

**INSTANCE:** Given a graph \( G \) with \( V \) nodes and \( E \) edges, and a positive integer \( K \leq ||V|| \).

**QUESTION:** Is \( G \) \( K \)-colorable, i.e., does there exist a function \( f : V \rightarrow \{1, 2, \ldots, K\} \) such that \( f(u) \neq f(v) \) whenever \( \{u, v\} \in E \)?

First, for our purpose, we must transform this problem into a binary CSP problem, e.g., by representing each node in the graph as a variable, each color as a value, and translating the edges in the graph into (binary) constraints. In accordance with Williams and Hogg [19], we consider a graph with \( \mu \) nodes, \( b \) colors and an average connectivity (i.e., the average number of edges incident on a node) \( \gamma \).

To use our model for backtracking on a CSP representation of this problem, we must establish the parameters \( N, M \) and \( p \). It is clear that \( N = \mu \) and \( M = b \), given the representation described above. In accordance with equation 1, the total number of possible binary constraint pairs is \( \binom{\mu}{2}b^2 \). With a graph connectivity \( \gamma \), the number of edges in the graph is \( \frac{1}{2}\gamma\mu \). Hence, the number of available binary constraint pairs \( B \) in the CSP representation is

\[
B = \left( \frac{\mu}{2} \right) - \frac{1}{2}\gamma\mu b^2 + \frac{1}{2}\gamma\mu(b^2 - b) \tag{44}
\]

In words: for each of the variable pairs connected by edges, \( b^2 - b \) value pairs are permitted (i.e., all pairs of different colors). For the variable pairs not connected by an edge, all \( b^2 \) value pairs are allowed. Using equation 2, we find for the graph coloring problem

\[
p = 1 - \frac{\gamma}{(\mu - 1)b} \tag{45}
\]

Next, we use our model to predict the critical connectivity \( \gamma_{\text{crit}} \) at which the phase transition takes place for the graph coloring problem. In accordance with the work by Crawford and Auton [2], we assume that this phase transition occurs when the probability of finding any solution is 0.5, i.e., the crossover point (see subsection 5.2). Hence, we use our model predictions to find the value of \( \gamma \) where \( p_{\text{crit}}(1) = 0.5 \), indicated by “(vL&vdH) \( \gamma_{\text{crit}} \)” (see table 1). The results by Williams and Hogg “(W&H) \( \gamma_{\text{crit}} \)” are copied from [19]. The “(C,K&T) \( \gamma_{\text{crit}} \)” in this table are measured values, taken from Cheeseman et al. [1]. Notice that there is a difference between our predictions and the predictions of Williams and Hogg. This difference is probably due to the fact that our model is based on the problem of finding a first solution, whereas Williams and Hogg base their model on the problem of finding all solutions. This conjecture is strengthened, if we look at the \( \gamma_{\text{crit}} \) we obtained if we assume the phase transition occurs where the number of solutions equals one. A value for this \( \gamma_{\text{crit}} \) can be computed using Haralick and Elliott’s model (equation (10 for \( p_{\text{trans}} \) together with equation (45)) and is illustrated in the column labeled “(H&E) \( \gamma_{\text{crit}} \)”.

32
In table 1 columns 3, 4, and 5 contain model predictions, and column 6 measured values. We remark that for graph coloring the predictions based on the crossover point are in better conformance with the measured values for $\gamma_{\text{crit}}$ than the predictions based on $p_{\text{trans}}$. This is in contrast with our results on random CSP problems (cf. figure 6).
In this article two useful models for CSP search processes for a first solution have been constructed viz. a model for standard backtracking and a model for min forward checking. They are based on the same assumptions as the model by Haralick and Elliott. The models predict two aspects of search behavior, viz. the probability that a solution will be found, and the average cost (consistency checks) involved in finding a first solution. The prediction of phase transition can be considered a special case. Below, we give the conclusions of our research.

First, our models form a valuable extension to the model by Haralick and Elliott for the case of a first solution, cf. subsection 5.1.

Second, the “strawman” prediction for a first solution is not accurate enough, especially not if $p \to 1$, cf. subsection 6.1, figures 8 and 9.

Third, both models correspond well to observed values for random CSP problems, cf. subsection 6.1. However, we have observed a discrepancy in accuracy in the region where a chance for a solution exists and the variation in costs is high (i.e., in part of the phase transition region), cf. subsection 6.1. Since this region will grow smaller for larger problems (cf. Cheeseman et al. [1]) we expect better agreement between theory and practice as problem size increases.

Fourth, from the measurements on non-random CSP problems, such as instances of the $N$-queens problem, we conclude that in this case (1) the model for standard backtracking does not conform to the measurements, and (2) the model for min forward checking provides an accurate prediction, cf. subsection 6.2. We expect that the discrepancies in the predictions by the model of standard backtracking are the result of the problem structure of the $N$-queens problem. This should be investigated in future research. Possible research extensions could be based on the work of Smith [14].

Fifth, as a special case, the models can be used to predict the phase transition. For this purpose it is necessary to obtain the relevant values for $N$, $M$, and $p$. At least for graph coloring problems, we conclude that the predictions are reasonably accurate.

From the conclusions above we infer that our model predictions for the average cost of a first solution show that for different values of the model parameters $N$, $M$, and $p$, different search algorithms are preferable (cf. subsection 5.2). From this observation we conclude that given a suitable CSP representation of a problem, our models can be used to select a search algorithm that is efficient when solving the problem.

In this article we have presented two models, viz. for standard backtracking and for min forward checking. We note that a model for (regular) forward checking amounts to a simplification of the model for min forward checking. Our future research aims at providing models for other interesting backtracking algorithms, e.g., informed backtracking [9]. Since we have indicated that such models can be constructed for different members of the class of backtracking algorithms, we assume it possible to construct a set of models, all predicting CSP search processes adequately. Using this set in practice, we can then choose the most appropriate algorithm for any given problem.
Acknowledgements

The authors wish to thank Jos Uiterwijk, Victor Allis, and Jacob Groote, for their discussions, ideas and helpful suggestions. Moreover they wish to thank Richard Korf for providing us with comments of persons who were interested in proofreading an early version of this article.

References


