Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems

Steven Minton[1] **Mark D. Johnston**[2] **Andrew B. Philips**[1] **Philip Laird**[3]

[1]Sterling Federal Systems
NASA Ames Research Center
AI Research Branch
Mail Stop: 269-2
Moffett Field, CA 94035 USA

[2]Space Telescope Science Institute
3700 San Martin Drive,
Baltimore, MD 21218 USA

[3]NASA Ames Research Center
AI Research Branch
Mail Stop: 269-2
Moffett Field, CA 94035 USA

Abbreviated Title: "Minimizing Conflicts: A Heuristic Repair Method"

Abstract

This paper describes a simple heuristic approach to solving large-scale constraint satisfaction and scheduling problems. In this approach one starts with an inconsistent assignment for a set of variables and searches through the space of possible repairs. The search can be guided by a value-ordering heuristic, the *min-conflicts heuristic*, that attempts to minimize the number of constraint violations after each step. The heuristic can be used with a variety of different search strategies.

We demonstrate empirically that on the n-queens problem, a technique based on this approach performs orders of magnitude better than traditional backtracking techniques. We also describe a scheduling application where the approach has been used successfully. A theoretical analysis is presented both to explain why this method works well on certain types of problems and to predict when it is likely to be most effective.

1 Introduction

One of the most promising general approaches for solving combinatorial search problems is to generate an initial, suboptimal solution and then to apply local *repair* heuristics[36, 28, 32, 30, 44, 38, 19]. Techniques based on this approach have met with empirical success on many combinatorial problems, including the traveling salesman and graph partitioning problems[20]. Such techniques also have a long tradition in AI, most notably in problem-solving systems that operate by debugging initial solutions [37, 40]. In this paper, we describe how this idea can be extended to constraint satisfaction problems (CSPs) in a natural manner.

Most of the previous work on CSP algorithms has assumed a "constructive" backtracking approach in which a partial assignment to the variables is incrementally extended. In contrast, our method creates a complete, but inconsistent assignment and then repairs constraint violations until a consistent assignment is achieved. The method is guided by a simple ordering heuristic for repairing constraint violations: identify a variable that is currently in conflict and select a new value that minimizes the number of outstanding constraint violations.

We present empirical evidence showing that on some standard problems our approach is considerably more efficient than traditional constructive backtracking methods. For example, on the n-queens problem, our method quickly finds solutions to the one million queens problem[30]. We argue that the reason that repair-based methods can outperform constructive methods is because a complete assignment can be more informative in guiding search than a partial assignment. However, the utility of the extra information is domain dependent. To help clarify the nature of this potential advantage, we present a theoretical analysis that describes how various problem characteristics may affect the performance of the method. This analysis shows, for example, how the "distance" between the current assignment and solution (in terms of the minimum number of repairs that are required) affects the expected utility of the heuristic.

The work described in this paper was inspired by a surprisingly effective neural network developed by Adorf and Johnston [2, 22] for scheduling astronomical observations on the Hubble Space Telescope. Our heuristic CSP method was distilled from an analysis of the network. In the process of carrying out the analysis, we discovered that the effectiveness of the network has little to do with its connectionist implementation. Furthermore, the ideas employed in the network can be implemented very efficiently within a symbolic CSP framework. The symbolic implementation is extremely simple. It also has the advantage that several different search strategies can be employed, although we have found that hill-climbing methods are particularly well-suited for the applications that we have investigated.

We begin the paper with a brief review of Adorf and Johnston's neural network, and then describe our symbolic method for heuristic repair. Following this, we describe empirical results with the n-queens problem, graph-colorability problems and the Hubble Space Telescope scheduling application. Finally, we consider a theoretical model identifying general problem characteristics that influence the performance of the method.

2 Previous Work: The GDS Network

By almost any measure, the Hubble Space Telescope scheduling problem is a complex task [21, 34, 43]. Between ten thousand and thirty thousand astronomical observations per year must be scheduled, subject to a great variety of constraints including power restrictions, observation priorities, time-dependent orbital characteristics, movement of astronomical bodies, stray light sources, etc. Because the telescope is an extremely valuable resource with a limited lifetime, efficient scheduling is a critical concern. An initial scheduling system, developed using traditional programming methods, highlighted the difficulty of the problem; it was estimated that it would take over three weeks for the system to schedule one week of observations. As described in section 4.2, this problem was remedied by the development of a successful constraint-based system to augment the initial system. At the heart of the constraint-based system is a neural network developed by Adorf and Johnston, the Guarded Discrete Stochastic (GDS) network, which searches for a schedule[2, 22].

From a computational point of view the network is interesting because Adorf and Johnston found that it performs well on a variety of tasks, in addition to the space telescope scheduling problem. For example, the network performs significantly better on the n-queens problem than methods that were previously developed. The n-queens problem requires placing n queens on an $n \times n$ chessboard so that no two queens share a row, column or diagonal. The network has been used to solve problems of up to 1024 queens, whereas most heuristic backtracking methods encounter difficulties with problems one-tenth that size[39].

The GDS network is a modified Hopfield network[18]. In a standard Hopfield network, all connections between neurons are symmetric. In the GDS network, the main network is coupled asymmetrically to an auxiliary network of *guard neurons* which restricts the configurations that the network can assume. This modification enables the network to rapidly find a solution for many problems, even when the network is simulated on a serial machine. Unfortunately, convergence to a stable configuration is no longer guaranteed. Thus the network can fall into a local minimum involving a group of unstable states among which it will oscillate. In practice, however, if the network fails to converge after some number of neuron state transitions, it can simply be stopped and started over.

To illustrate the network architecture and updating scheme, let us consider how the network is used to solve binary constraint satisfaction problems. A problem consists of n variables, $X_1 \ldots X_n$, with domains $D_1 \ldots D_n$, and a set of binary constraints. Each constraint $C_\alpha(X_j, X_k)$ is a subset of $D_j \times D_k$ specifying incompatible values for a pair of variables. The goal is to find an assignment for each of the variables which satisfies the constraints. (In this paper we only consider the task of finding a single solution, rather than that of finding all solutions.) To solve a CSP using the network, each variable is represented by a separate set of neurons, one neuron for each of the variable's possible values. Each neuron is either "on" or "off", and in a solution state, every variable will have exactly one of its corresponding neurons "on", representing the value of that variable. Constraints are represented by inhibitory (i.e., negatively weighted) connections between the neurons. To insure that every variable is assigned a value, there is a guard neuron for each set of neurons representing a variable; if no neuron in the set is on, the guard neuron will provide an excitatory input that is large enough to turn one on. (Because of the way the connection weights are set up, it is unlikely that the guard neuron will turn on more than one neuron.) The network is updated on each cycle by randomly picking a set of neurons that represents a variable, and flipping the state of the neuron in that set whose input is *most inconsistent* with its current output (if any). When all neurons' states are consistent with their input, a solution is achieved.

To solve the n-queens problem, for example, each of the $n \times n$ board positions is represented by a neuron whose output is either one or zero depending on whether a queen is currently placed in that position or not. (Note that this is a local representation rather than a distributed representation of the board.) If two board positions are inconsistent, then an inhibiting connection exists between the corresponding two neurons. For example, all the neurons in a column will inhibit each other, representing the constraint that two queens cannot be in the same column. For each row, there is a guard neuron connected to each of the neurons in that row which gives the neurons in the row a large excitatory input, enough so that at least one neuron in the row will turn on. The guard neurons thus enforce the constraint that one queen in each row must be on. As described above, the network is updated on each cycle by randomly picking a row and flipping the state of the neuron in that row whose input is most inconsistent with its current output. A solution is realized when the output of every neuron is consistent with its input.

3 Why does the GDS Network Perform So Well?

Our analysis of the GDS network was motivated by the following question: "Why does the network perform so much better than traditional backtracking methods on certain tasks"? In particular, we were intrigued by the results on the n-queens problem, since this problem has received considerable attention from previous researchers. For n-queens, Adorf and Johnston found empirically that the network requires a linear number of transitions to converge. Since each transition requires linear time, the expected (empirical) time for the network to find a solution is $O(n^2)$. To check this behavior, Johnston and Adorf ran experiments with n as

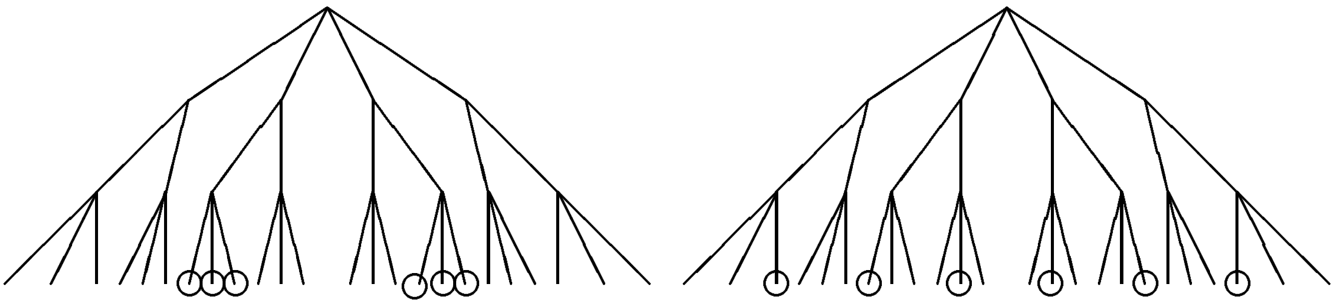

Figure 1: Solutions Clustered vs. Solutions Evenly Distributed

high as 1024, at which point memory limitations became a problem. [1]

3.1 Nonsystematic Search Hypothesis

Initially, we hypothesized that the network's advantage came from the nonsystematic nature of its search, as compared to the systematic organization inherent in depth-first backtracking. There are two potential problems associated with systematic depth-first search. First, the search space may be organized in such a way that poorer choices are explored first at each branch point. For instance, in the n-queens problem, depth-first search tends to find a solution more quickly when the first queen is placed in the center of the first row rather than in the corner; apparently this occurs because there are more solutions with the queen in the center than with the queen in the corner [39]. Nevertheless, most naive algorithms tend to start in the corner simply because humans find it more natural to program that way. However, this fact by itself does not explain why nonsystematic search would work so well for n-queens. A backtracking program that randomly orders rows (and columns within rows) performs much better than the naive method, but still performs poorly relative to the GDS network.

The second potential problem with depth-first search is more significant and more subtle. As illustrated by figure 1, a depth-first search can be a disadvantage when solutions are not evenly distributed throughout the search space. In the tree at the left of the figure, the solutions are clustered together. In the tree on the right, the solutions are more evenly distributed. Thus, the average distance between solutions is greater in the left tree. In a depth-first search, the average time to find the first solution increases with the average distance between solutions. Consequently depth-first search performs relatively poorly in a tree where the solutions are clustered, such as that on the left [13, 29]. In comparison, a search strategy which examines the leaves of the tree in random order is unaffected by solution clustering.

We investigated whether this phenomenon explained the relatively poor performance of depth-first search on n-queens by experimenting with a randomized search algorithm, called a Las Vegas algorithm [5]. The algorithm begins by selecting a path from the root to a leaf. To select a path, the algorithm starts at the root node and chooses one of its children with equal probability. This process continues recursively until a leaf is encountered. If the leaf is a solution the algorithm terminates, if not, it starts over again at the root and selects a path. The same path may be examined more than once, since no memory is maintained between successive trials.

The Las Vegas algorithm does, in fact, perform better than simple depth-first search on n-queens. In

[1] The network, which is programmed in Lisp, requires approximately 11 minutes to solve the 1024 queens problem on a TI Explorer II. For larger problems, memory becomes a limiting factor because the network requires approximately $O(n^2)$ space. (Although the number of connections is actually $O(n^3)$, some connections are computed dynamically rather than stored).

fact, this result was already known [5]. However, the performance of the Las Vegas algorithm is still not nearly as good as that of the GDS network, and so we concluded that the systematicity hypothesis alone cannot explain the network's behavior.

3.2 Informedness Hypothesis

Our second hypothesis was that the network's search process uses information about the current assignment that is not available to a constructive backtracking program. We now believe this hypothesis is correct, in that it explains why the network works so well. In particular, the key to the network's performance appears to be that state transitions are made so as to reduce the number of outstanding inconsistencies in the network; specifically, each state transition involves flipping the neuron whose output is most inconsistent with its current input. From a constraint satisfaction perspective, it is as if the network reassigns a value for a variable by choosing the value that violates the fewest constraints. This idea is captured by the following heuristic:

> **Min-Conflicts heuristic:**
> *Given:* A set of variables, a set of binary constraints, and an assignment specifying a value for each variable. Two variables *conflict* if their values violate a constraint.
> *Procedure:* Select a variable that is in conflict, and assign it a value that minimizes the number of conflicts.[2] (Break ties randomly.)

We have found that the network's behavior can be approximated by a symbolic system that uses the min-conflicts heuristic for hill climbing. The hill-climbing system starts with an initial assignment generated in a preprocessing phase. At each choice point, the heuristic chooses a variable that is currently in conflict and reassigns its value, until a solution is found. The system thus searches the space of possible assignments, favoring assignments with fewer total conflicts. Of course, the hill-climbing system can become "stuck" in a local maximum, in the same way that the network may become "stuck" in a local minimum. In the next section we present empirical evidence to support our claim that the min-conflicts approach can account for the network's effectiveness.

There are two aspects of the min-conflicts hill-climbing method that distinguish it from standard CSP algorithms. First, instead of incrementally constructing a consistent partial assignment, the min-conflicts method *repairs* a complete but inconsistent assignment by reducing inconsistencies. Thus, it uses information about the current assignment to guide its search that is not available to a standard backtracking algorithm. Second, the use of a hill-climbing strategy rather than a backtracking strategy produces a different style of search.

Extracting the method from the network enables us to tease apart and experiment with its different components. In particular, the idea of repairing an inconsistent assignment can be used with a variety of different search strategies in addition to hill climbing. For example, we can backtrack through the space of possible repairs, rather than using a hill-climbing strategy, as follows. Given an initial assignment generated in a preprocessing phase, we can employ the min-conflicts heuristic to order the choice of variables and values to consider, as described in figure 2. Initially, the variables are all on a list of VARS-LEFT, and as they are repaired, they are pushed onto a list of VARS-DONE. The algorithm attempts to find a sequence of repairs, such that no variable is repaired more than once. If there is no way to repair a variable in VARS-LEFT without violating a previously repaired variable (a variable in VARS-DONE), the algorithm backtracks.

Notice that this algorithm is simply a standard backtracking algorithm augmented with the min-conflicts heuristic to order its choice of which variable and value to attend to. This illustrates an important point. The backtracking repair algorithm incrementally extends a consistent partial assignment (i.e., VARS-DONE), as does a constructive backtracking program, but in addition, uses information from the initial assignment

[2] In general, the heuristic attempts to minimize the number of other variables that will need to be repaired. For binary CSPs, this corresponds to minimizing the number of conflicting variables. For general CSPs, where a single constraint may involve several variables, the exact method of counting the number of variables that will need to be repaired depends on the particular constraint. The space telescope scheduling problem is a general CSP, whereas the other tasks described in this paper are binary CSPs.

```
 Procedure INFORMED-BACKTRACK (VARS-LEFT VARS-DONE)
  If all variables are consistent, then solution found, STOP.
  Let VAR = a variable in VARS-LEFT that is in conflict.
  Remove VAR from VARS-LEFT.
  Push VAR onto VARS-DONE.
  Let VALUES = list of possible values for VAR ordered in ascending order according to number of
               conflicts with variables in VARS-LEFT.
  For each VALUE in VALUES, until solution found:
    If VALUE does not conflict with any variable that is in VARS-DONE,
    then Assign VALUE to VAR.
         Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
    end if
  end for
 end procedure

Begin program
 Let VARS-LEFT = list of all variables, each assigned an initial value.
 Let VARS-DONE = nil
 Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
End program
```

Figure 2: Informed Backtracking Using the Min-Conflicts Heuristic

(i.e., VARS-LEFT) to bias its search. Thus, it is a type of *informed backtracking.* We still characterize it as repair-based method since its search is guided by a complete, inconsistent assignment.

4 Experimental Results

In this section we evaluate the performance of the min-conflicts heuristic on some standard tasks. These experiments identify problems on which min-conflicts performs well, as well as problems on which it performs poorly. The experiments also show the extent to which the min-conflicts approach approximates the behavior of the GDS network.

Our experiments focus on the two search strategies described in the previous section, the hill-climbing repair strategy and the backtracking repair strategy. These strategies provide a starting point for our analysis, although many more sophisticated search strategies exist. In general, these two strategies have the following advantages and disadvantages:

1. Hill climbing: This strategy most closely replicates the behavior of the GDS network. The disadvantage is that a hill-climbing program can get caught in local maxima, in which case it will not terminate.

2. Informed backtracking: As described earlier, this strategy is a standard backtracking strategy augmented with the min-conflicts heuristic for ordering the assignment of variables and values; this can be viewed as backtracking in the space of possible repairs. The advantage of this strategy is that it is complete – if there is a solution, it will eventually be found; if not, failure will be reported. Unfortunately, this is of limited significance for large-scale problems because terminating in a failure can take a very long time.

4.1 The N-Queens Problem

The n-queens problem, originally posed in the 19th century, has become a standard benchmark for testing CSP algorithms. In a sense, the problem of finding a single solution has been solved, since there are a number of analytic methods which yield a solution in linear time [1]. For example, there are certain well-known patterns that can be instantiated to produce a solution. Nevertheless, the problem has been perceived as relatively "hard" for heuristic search methods. Several studies of the n-queens problem [39, 15, 25] have compared heuristic backtracking methods such as search rearrangement backtracking (e.g., most-constrained first), forward checking, dependency-directed backtracking, etc. To the best of our knowledge, the GDS network was the first search method which could consistently solve problems involving hundreds of queens in several minutes.

On the n-queens problem, Adorf and Johnston [2] reported that the probability of the GDS network converging increases with the size of the problem. For large problems, e.g., $n > 100$ (where n is the number of queens), they observed that the network almost always converges. Moreover, the median number of transitions required for convergence is only about $1.16n$. Since it takes $O(n)$ time to execute a transition (i.e., picking a neuron and updating its connections), the expected time to solve a problem is (empirically) $O(n^2)$.

n	conflicts after initialization
$n = 10^1$	3.11
$n = 10^2$	7.35
$n = 10^3$	9.75
$n = 10^4$	10.96
$n = 10^5$	12.02
$n = 10^6$	12.80

Table 1: Number of Conflicts After Initialization

To compare the network with our min-conflicts approach, we constructed a hill-climbing program that operates as follows. A preprocessing phase creates an initial assignment using a greedy algorithm that iterates through the rows, placing each queen on the column where it conflicts with the fewest previously placed queens (breaking ties randomly). In the subsequent repair phase the program keeps repairing the assignment until a solution is found. To make a repair, the program selects a queen that is in conflict and moves it to the column (within the same row) where it conflicts with the fewest other queens (breaking ties randomly). A repair can be accomplished in $O(n)$ time by maintaining a list of the queens currently in conflict and an array of counters indicating the number of conflicts in each column and diagonal.

	Constructive		Repair-based	
N	Standard Backtrack	Most Constrained Backtrack	MinConflicts Hill Climbing	MinConflicts Backtrack
$n = 10^1$	53.8	17.4	57.0	46.8
$n = 10^2$	4473 (70%)	687 (96%)	55.6	25.0
$n = 10^3$	88650 (13%)	22150 (81%)	48.8	30.7
$n = 10^4$	*	*	48.5	27.5
$n = 10^5$	*	*	52.8	27.8
$n = 10^6$	*	*	48.3	26.4

$*$ = exceeded computational resources

Table 2: Number of Backtracks/Repairs for N-Queens Algorithms

Interestingly, in our initial experiments we found that the hill-climbing program performs significantly *better* than the network. For $n \geq 100$ the program has never failed to find a solution. Moreover, the required number of repairs appears to remain *constant* as n increases. For comparison, recall that the required number of repairs for the network increases linearly with n. After further analysis, we found that this discrepancy can be accounted for by the network's and the hill-climbing program's different initialization processes. In particular, whereas the network starts with no queens assigned in the initial state, the hill-climbing program's preprocessing phase invariably produces an initial assignment that is "close" to a solution. As shown in Table 1, the number of conflicting queens in the initial assignment grows extremely slowly, from a mean of 3.1 for $n = 10$ to a mean of 12.8 for $n = 10^6$. We found that if we start the network in an initial state produced by our preprocessing algorithm, the network and the hill-climbing program perform comparably. (We note, however, that the network requires $O(n^2)$ space, as compared to the $O(n)$ space required by the hill-climbing program, which prevented us from running very large problems on the network.) On the other hand, if we start the the hill-climbing program with a random initial assignment, the required number of repairs tends to grow linearly. This is not surprising, since the number of conflicts in a random initialization also tends to grow linearly.

Table 2 compares the efficiency of our hill-climbing program and several backtracking programs. Each program was run one hundred times for n increasing from ten to one million. Each entry in the table shows the mean number of queens moved, where each move is either a backtrack or a repair, depending on the program. A bound of $n \times 100$ queen movements was employed so that the experiments could be conducted in a reasonable amount of time; if the program did not find a solution after moving $n \times 100$ queens, it was terminated and credited with $n \times 100$ queen movements. For the cases when this occurred, the corresponding table entry indicates in parentheses the percentage of times the program completed successfully. The first column shows the results for a standard constructive backtracking program. For $n \geq 1000$, the program was ineffective. The second column in the table shows the results for informed backtracking using the "most-constrained first" heuristic. This program is a constructive backtracking program that selects the row that is most constrained when choosing the next row on which to place a queen. In an empirical study of the n-queens problem, Stone and Stone [39] found that this was by far the most powerful heuristic for the n-queens problem out of several described earlier by Bitner and Reingold[4]. The program exhibited highly variable behavior. At $n = 1000$, the program found a solution on only 81% of the runs, but three-quarters of these successful runs required fewer than 100 backtracks. Unfortunately, for $n > 1000$, one hundred runs of the program required considerably more than 12 hours on a SPARCstation1, both because the mean number of backtracks grows rapidly and because the "most-constrained first" heuristic takes $O(n)$ time to select the next row after each backtrack. Thus we were prevented from generating sufficient data for $n > 1000$. The next column in the table shows the results for hill climbing using the min-conflicts heuristic. As discussed above, this algorithm performed extremely well, requiring only about 50 repairs irrespective of problem size. The final column shows the results for an informed backtracking program that uses the min-conflicts heuristic, backtracking within the space of possible repairs as described in the previous section. We augmented this program with a pruning heuristic that would initiate backtracking when the number of constraint violations along a path began to increase significantly. However, for $n \geq 100$, this program never backtracked (i.e., no queen had to be repaired more than once). This last program performs better than the hill-climbing program (although there is little room for improvement) primarily because the hill-climbing program may move the same queen repeatedly, which degrades its performance.

A disadvantage of the min-conflicts heuristic is that the time to accomplish a repair grows with the size of the problem. For n-queens, as noted above, each repair requires $O(n)$ time in the worst case. Of course, most heuristic methods require time to determine the best alternative at a choice point. For example, the "most-constrained" heuristic also requires $O(n)$ time at each choice point. However, with min-conflicts the tradeoff is clearly cost effective, at least for n-queens. Since the number of repairs remains approximately constant as n grows, the program's runtime is approximately linear. This is illustrated by figure 3, which shows the average runtime for the hill-climbing program. In terms of realtime performance, this program solves the million queens problem in less than four minutes on a SPARCstation1.

The cost of making a repair can be optimized for large problems, in which case the average solution time for the million-queens problem is reduced to less than a minute and a half. The program maintains a list of queens that are in conflict, as well as three arrays of counters indicating the number of queens in each column, row and diagonal. Rather than scanning a row for the position with the fewest conflicts, the optimized program maintains a list of empty columns (which tends to be quite small); it first checks for a zero-conflict position by looking for an empty column with no conflicts along the diagonals. If there is no zero-conflict position, the program repeatedly looks for a position with one conflict by randomly selecting a position and checking the number of conflicts in that position. Since there tend to be many positions with one conflict, this technique tends to succeed after just a few tries, so the total number of positions examined is generally very low.

Figure 3: Mean Solution Time for Hill-Climbing Program on N-Queens Problem

One obvious conclusion from these results is that n-queens is actually a very easy problem given the right method. Interesting, two other heuristic methods that can quickly solve n-queens problems have also recently been invented. (By coincidence, these two other methods and our method were all developed and published independently.) While both methods are specific to n-queens, one method is a repair-based method that is similar to ours in spirit[38], whereas the other employs a constructive backtracking approach[23]. This latter method uses a combination of variable and value-ordering heuristics which take advantage of the particular structure inherent in n-queens. This shows that one *can* solve n-queens problems quickly with a traditional, constructive backtracking method. Nevertheless, given the comparative simplicity of our method, it would seem that n-queens is more naturally solved using a repair-based approach.

4.2 Scheduling Applications

Whereas the n-queens problem is only of theoretical interest, scheduling algorithms have many practical applications. A scheduling problem involves placing a set of tasks on a time line, subject to temporal constraints, resource constraints, preferences, etc. The Hubble Space Telescope scheduling problem can be considered a constrained optimization problem[12, 10] where we must maximize both the number and the importance of the constraints that are satisfied. As noted earlier, the initial scheduling system developed for this application had difficulty producing schedules efficiently. The constraint-based system, SPIKE, that was developed to augment (and partially replace) the initial system has performed quite well using a relatively simple approach.

In part, the HST scheduling problem was made more tractable by dividing it into two parts, a long-term scheduling problem and a short-term scheduling problem. Currently SPIKE handles only the long-term problem. The long-term problem involves assigning approximately one year's worth of exposures to a set of "bins" or time segments of several days length. (The short-term problem involves deriving a detailed series of commands for the telescope and is addressed using different techniques [34].) The input to SPIKE is a set of detailed specifications for exposures that are to be scheduled on the telescope. The constraints relevant to the long term problem are primarily temporal constraints. As outlined in [21], some exposures are designed as calibrations or target acquisitions for others, and so must proceed them. Some must be executed at specific times, or at specific phases in the case of periodic phenomena. Some observations must be made at regular intervals, or grouped within a specified time span. The constraints vary in their importance; they range from "hard" constraints that cannot be violated under any circumstances, to "soft" constraints that represent good operating practices and scheduling goals.

SPIKE operates by taking the exposure specifications prepared by astronomers and compiling them into a set of tasks to be scheduled and a set of constraints on those tasks. Among other things, the compilation process takes the transitive closure of temporal constraints and explicitly represents each inferred constraint. For example, if TaskA must be before TaskB, and TaskB must be before TaskC, then the system will explicitly represent the fact that TaskA must be before TaskC as well. This explicit representation enables the scheduler to obtain a more accurate assessment of the number of conflicts in a given schedule.

In searching for a schedule, the GDS network follows the constraint satisfaction approach outlined in section 2. In effect, if a task is currently in conflict then it is removed from the schedule, and if a task is currently unscheduled then the network schedules it for the time segment that has the fewest constraint violations. However, the network uses only the hard constraints in determining the time segment with the fewest violations. Soft constraints are consulted when there are two or more "least conflicted" places to move a task.

The min-conflicts hill-climbing method has been shown to be as effective as the GDS network on representative data sets used for testing SPIKE, and it has been incorporated into the SPIKE system. One advantage in using the min-conflicts method, as compared to the GDS network, is that much of the overhead of using the network can be eliminated (particularly the space overhead). Moreover, because the min-conflicts heuristic is so simple, the min-conflicts scheduler was quickly coded in C and is extremely efficient. (The min-conflicts scheduler runs about an order of magnitude faster than the network, although some of the improvement is due to factors such as programming language differences, making a precise comparison difficult.) While this may be regarded as just an implementation issue, we believe that the clear and simple formulation of the method was a significant enabling factor. In addition, the simplicity of the method makes is easy to experiment with various modifications to the heuristic and the search strategy. This has significant practical import, since SPIKE is currently being used on other types of telescope scheduling problems where a certain amount of modification and tuning is required.

In general, scheduling appears to be an excellent application area for repair-based methods. Supporting evidence comes from previous work on other real-world scheduling applications by Zweben et al.[44], Biefeld and Cooper[3] and Kurtzmann[27]. Each of these projects use iterative improvement methods which can be characterized as repair-based. There are several reasons why repair-based methods are well-suited to scheduling applications. First, as Zweben and Gargan[45] have pointed out, unexpected events may require schedule revision, in which case dynamic rescheduling is an important issue. Repair-based methods can be used for rescheduling in a natural manner. Second, most scheduling applications involve optimization, at least to some degree, and repair-based methods are also naturally extended to deal with such issues. For example, in scheduling the Hubble Space Telescope, the goal is to maximize the amount of observing time and the priority of the chosen observations. The telescope is expected to remain highly over-subscribed, in that many more proposals will be submitted than can be accommodated by any schedule. On such problems, repair-based methods offer an alternative to traditional branch-and-bound techniques. Finally, as Biefeld and Cooper[3] have pointed out, there are real-world scheduling problems where humans find repair-based methods very natural. For example, human schedulers at JPL employ repair-based methods when constructing mission schedules for robotic spacecraft. For such problems, it may be relatively easy for people using a repair-based system to understand the system's solution and how it was arrived at.

4.3 Graph Coloring

In addition to n-queens problem and HST scheduling, Adorf and Johnston also tested the GDS network on graph 3-colorability problems. A graph 3-colorability problem consists of an undirected graph with n vertices. Each vertex must be assigned one of three colors subject to the constraint that no neighboring vertex is assigned the same color. Graph 3-colorability is a well-studied NP-complete problem that is used to model certain types of scheduling and resource allocation problems, such as examination scheduling and register allocation.

Adorf and Johnston found that the performance of the network depended greatly on the connectivity of the graph. On densely-connected graphs the network converged rapidly to a solution, while on sparsely-connected graphs the network performed much more poorly. We have repeated Adorf and Johnston's experiments using the min-conflicts approach, and found similar results. We have also found that there is a simple, well-known backtracking algorithm for coloring graphs that performs much better than either the network or any of our min-conflicts algorithms on sparsely-connected graphs. This provides a useful case for comparative analysis.

We used the same procedure for generating test problems as Adorf and Johnston. Solvable problems with n nodes and m arcs are generated as follows:

1. Create three groups of nodes, each with $n/3$ nodes.
2. Randomly create m arcs between nodes in different groups.
3. Accept the graph if it has no unconnected components.

Johnston and Adorf experimented with two classes of problem instances; one set with $m = 2n$ (i.e., average vertex degree of 4) and another with $m = n(n-1)/4$. We will refer to the former as the sparsely-connected graphs, and the latter as the densely-connected graphs.

Figure 4 compares the results published by Adorf and Johnston with our results. In Adorf and Johnston's experiments, graphs were tested in the range from $n = 30$ to $n = 180$. For each of the two types of graphs, three different instances of each size were generated, and the network was run 3000 times per graph. Our experiments with the min-conflicts hill-climbing algorithm employed the same experimental design.

Because the network is started with all nodes "uncolored", we employed a similar approach with the hill-climbing program so that the comparison would be fair. Thus, in the initialization phase, each vertex is labeled as "uncolored". An uncolored node is defined to conflict with each of its neighbors, regardless of their color.

The results demonstrate that the hill-climbing algorithm behaves similarly to the GDS network on both types of problems. This supports our hypothesis that the hill-climbing algorithm captures the essential characteristics of the network. As shown in figure 4a, the densely-connected graphs are easy to solve. Both methods tend to converge rather quickly on average. In particular, the median number of transitions required for convergence grows linearly with n. The sparsely-connected graphs are much harder. In these experiments, the network was given a bound of $9n$ transitions, after which the run was terminated. (The bound was chosen arbitrarily, but means in principle that each of the $3n$ neurons in the main network can transition three times.) The hill-climbing algorithm was therefore given a bound of $9n$ repairs. As illustrated in figure 4b, for both methods, the probability of success appears to decline exponentially with n. [3] Adorf and Johnston observed that as the number of nodes increases, it is highly likely that the network

[3] The use of an identical bound for both programs may give the hill-climbing algorithm a slight advantage. The GDS network requires separate transitions to deassign a variable and to assign a new value. In the hill-climbing program a single repair, in effect, simulates two transitions by the network (unless an initial "uncolored" value is being repaired). Additional experimentation has revealed that this advantage is relatively small, however. In fact, figure 4b shows that on the sparse graphs, the hill-climbing program performed a bit worse than the network for small n, although the significance of this is unclear due to the relatively large statistical variation in the difficulty of the smaller problems. Unfortunately, the original problems are unavailable and the network is no longer running, so additional experiments cannot be run.

Figure 4: Comparing the GDS network to Min-Conflicts Hill Climbing on Dense and Sparse Graph-Coloring Problems

will become caught in a local minimum in which a small number of neurons transition repeatedly. That is, the network becomes trapped, vacillating between several states. The hill-climbing algorithm behaves in a similar manner.

To determine whether the min-conflicts approach would be practical for graph-coloring applications, we compared our two min-conflicts algorithms to a simple constructive backtracking algorithm that is known to perform well on graph-coloring problems. The algorithm, originally proposed by Brelaz[6, 41], can be described as the repeated application of the following rule for choosing a node to color: colorings with maximum degree in the uncolored subgraph. Break ties randomly.

> Find the uncolored node that has the fewest consistent colorings with its neighbors. If there is more than one, then choose one that has the maximum degree in the uncolored subgraph. Break ties randomly.

Essentially, this is a variable ordering rule consisting of two criteria. The first criterion is a preference for the "most-constrained" variable. The tie-breaking criterion is a preference for the "most-constraining" variable. Thus, this rule is composed of two generic variable-ordering heuristics. No value-ordering heuristic is required.

The rule can be incorporated in a standard backtracking algorithm in the obvious manner. Turner [41] has shown that this algorithm will optimally color "almost all" random k-colorable graphs without backtracking. This result actually says more about the distribution of random k-colorable graphs than about the effectiveness of the algorithm, but nonetheless, the Brelaz algorithm outperforms other algorithms we have tried.

For a fair comparison between the Brelaz algorithm and our two min-conflicts algorithms, a good initialization method for the min-conflicts algorithms is presumably required. We can use the Brelaz rule itself to arrive at an initialization for our min-conflicts algorithms. Specifically, the initialization process makes one pass through the vertices of the graph, using the Brelaz variable ordering rule to pick the next vertex to color. If no color consistent with the node's neighbors is available, a color is chosen that minimizes the number of conflicts. This process results in initial colorings with many fewer conflicts than random colorings. Table 3 shows the percentage of times that the initialization routine, by itself, finds a solution, for graphs of size n. Each entry in the table is based on 100 runs of the initialization routine for eight problems of size n. to the sparsely-connected and densely-connected graphs described computed problems.

Since the initialization process consistently finds solutions for the easy densely-connected graphs (eliminating the need for a repair phase), we restricted our experiments to the hard sparsely-connected graphs. Figure 5 compares the performance of the Brelaz algorithm with min-conflicts hill climbing. For completeness, the figure also shows a third algorithm, an informed backtracking problem that uses min-conflicts to search through the space of repairs. For each method, we tested eight randomly generated problems of size n,

n	Sparse Graphs	Dense Graphs
30	63.19%	100.00%
60	50.13%	100.00%
90	40.37%	100.00%
120	32.75%	100.00%
150	32.87%	100.00%
180	23.75%	100.00%

Table 3: Probability that Initialization Alone will Solve the Problem

Figure 5: Comparing Brelaz Backtracking with Two Min-Conflicts Methods

for 100 runs per problem. The graph shows the probability of finding a solution within $9n$ repairs/backtracks. (The results do not include trials where no repairs were required, or where Brelaz found the solution without backtracking. This is fair since the two repair-based methods use the Brelaz rule for initialization.)

The conclusion from this experiment is that the Brelaz backtracking algorithm obviously outperforms both of the min-conflicts methods. Of the two latter methods, informed backtracking performs slightly better. In addition, comparing the performance of hill climbing with and without the Brelaz initialization method (figure 5 and figure 4) shows that the initialization method improves performance, but not dramatically.[4]

The experiments also demonstrate clearly that sparse graphs are much harder to color than dense graphs, for both the Brelaz method as well as for the min-conflicts methods. Intuitively, the reason that dense graphs are easy to color is that they are so overconstrained that a mistake is both unlikely and easily corrected. For min-conflicts, a mistake is easily corrected because the choice of color at a vertex is greatly influenced by the colors of all of its neighbors. For the Brelaz backtracking method, a mistake is easily corrected since the subsequent choices will be pruned quickly due to the overconstrained nature of the problem. In a study motivated in part by these experiments, Cheeseman et al. [7] have shown that as the average connectivity of a (connected) graph increases, a "phase transition" occurs, and it is at this point that most of the hard graph colorability problems are found. In other words, since a constraint satisfaction problem is easy if it is either underconstrained or overconstrained, hard problems can be expected to lie within the boundary between overconstrained and underconstrained problems. Our sparsely-connected graphs lie within this boundary area.

Figure 6 illustrates how the difficulty of sparsely-connected connected graphs manifests itself for min-conflicts. The group of nodes on the left of the graph represents one consistent coloring, and the group on the right represents a different consistent coloring. But the two colorings are inconsistent with each other.

[4] Interestingly, the Brelaz initialization method actually degrades performance on the smallest graphs (where $n = 30$). This is an anomaly which we cannot as yet explain.

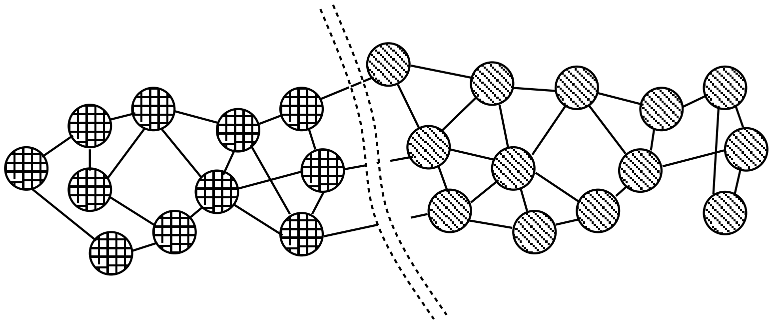

Figure 6: An Unlucky Initialization

This situation can frequently arises as a result of the initialization process. On the surface, the assignment would appear to be a good one, since there are at most three pairs of nodes in conflict. However, to achieve a solution, the boundary between the consistent colorings must be "pushed" completely to the left or right during the repair phase. Unfortunately, in this situation, there is not enough information locally available to direct min-conflicts. We have observed, in animations of the hill-climbing program, that the boundary tends to vacillate back and forth with little overall progress being made.

The excellent performance of the Brelaz algorithm led us to experiment with backtracking repair algorithms that are a hybrid of Brelaz and min-conflicts. The best hybrid algorithm we found first employs the Brelaz initialization routine described above. Then a modified version of the Brelaz variable selection rule is used:

> Of the nodes that have not yet been repaired, find the node that has the fewest consistent colorings with its already-repaired neighbors. If there is more than one, then choose one that is in conflict with a previously repaired node. If there is still more than one candidate, choose the one with the maximum degree in the unrepaired subgraph.

The hybrid algorithm uses this rule for variable ordering and the min-conflicts heuristic for value ordering. Interestingly, once the initial assignment is made, this algorithm has a higher probability of finding a solution without backtracking than Brelaz. On the other hand, when the algorithm does backtrack, it tends to require more backtracking on average than Brelaz, probably because it does not make as effective use of the "most constraining" criteria for variable selection. Unfortunately, the total time required by the hybrid algorithm tends to increase faster than the total time required by Brelaz, and thus the hybrid method appears to be primarily of academic interest.

4.4 Summary of Experimental Results

For each of the three tasks we have examined in detail, n-queens, HST scheduling and graph 3-colorability, we have found that the GDS network's behavior can be approximated by the min-conflicts hill-climbing algorithm. To this extent, we have a theory that explains the network's behavior. Obviously, there are certain practical advantages to having "extracted" this method from the network. First, the method is very simple, and so can be programmed extremely efficiently, especially if done in a task-specific manner. Second, the heuristic we have identified, that is, choosing the repair which minimizes the number of conflicts, is very general. It can be used in combination with different search strategies and task-specific heuristics, an important factor for most practical applications.

For example, the min-conflicts heuristic can be used in combination with a variety of variable ordering heuristics. In the previous section, for instance, we described a hybrid program in which the Brelaz variable ordering heuristic is adapted for use with min-conflicts value-ordering heuristic. We have also experimented with a hill-climbing program that uses "max-conflicts" as a variable ordering heuristic in conjunction with the min-conflicts value ordering heuristic. On graph-coloring problems, the resulting program tends to outperform min-conflicts alone, although performance is still not as good as the Brelaz algorithm.

Insofar as the power of our approach is concerned, our experimental results are encouraging. We have identified two tasks, n-queens and HST scheduling, which appear more amenable to our repair-based approach than the traditional constructive approach that incrementally extends a consistent partial assignment. This is not to say that a repair-based approach will do better than *any* constructive approach on these tasks, but merely that our simple, repair-based approach has done relatively well in comparison to the obvious constructive strategies we tried. We also note that repair-based methods have a special advantage for scheduling tasks, since they can be used for overconstrained problems and for rescheduling problems in a natural manner. Thus it seems likely that there are other applications for which our approach will prove useful.

5 Analysis

The previous section showed that, compared to constructive approaches, our repair-based approach is extremely effective on some tasks, such as placing queens on a chessboard, and less effective on other tasks, such as coloring sparsely-connected graphs. We claimed that the min-conflicts heuristic takes advantage of information in the complete assignment to guide its search; this information is not available to a constructive backtracking algorithm that incrementally extends a partial assignment. Thus the advantage of the min-conflicts heuristic over constructive approaches depends on how "useful" this information is. In this section we formalize this intuition. Specifically, we investigate how the use of a complete assignment informs the choice of which value to pick. The analysis reveals how the effectiveness of the min-conflicts heuristic is influenced by various characteristics of a task domain. The analysis is independent of any particular search strategy, such as hill climbing or backtracking.

5.1 Modeling the Min-Conflicts Heuristic

Consider a constraint satisfaction problem with n variables, where each variable has k possible values. We restrict our consideration to a simplified model where every variable is subject to exactly c binary constraints, and we assume that there is only a single solution to the problem, that is, exactly one satisfying assignment. We address the following question: What is the probability that the min-conflicts heuristic will make a mistake when it assigns a value to a variable that is in conflict? We define a mistake as choosing an incorrect value that will have to be changed before the solution is found. We note that for our informed backtracking program a mistake of this sort may prove quite costly, since an entire subtree must be explored before another value can be assigned.

For any assignment of values to the variables, there is a set of d variables whose values must be changed to convert the assignment into the solution. We can regard d as a measure of distance to the solution. The key to our analysis is the following observation. Given a variable V to be repaired, only one of its k possible values will be correct[5] and the other $k-1$ values will be incorrect (i.e., mistakes). Whereas the correct value may conflict with at most d other variables in the assignment, an incorrect value may conflict with as many as c other variables. Thus, as d shrinks, the min-conflicts heuristic should be less likely to make a mistake when it repairs V. In fact, if each of the $k-1$ incorrect values has more than d conflicts, then the min-conflicts heuristic cannot make a mistake – it will select the correct value when it repairs this variable, since the correct value will have fewer conflicts than any incorrect value.

We can use this idea to bound the probability that the min-conflicts heuristic will make a mistake when repairing variable V. Let V' be a variable related to V by a constraint. We assume that an incorrect

[5] Although a variable is in conflict, its assigned value may actually be the correct value. This can happen when the variable with which it conflicts has an incorrect value. In this paper we have defined the min-conflicts heuristic so that it can choose *any* possible value for the variable, including the variable's current value.

value for V conflicts with an arbitrary value for V' with probability p, independent of the variables V and V'. Consider an arbitrary incorrect value for V. Let N_b be the total the number of conflicts between this incorrect value and the assigned values for the other variables. Given the above assumptions, the expected value of N_b is pc, because there are exactly c variables that share a constraint with V, and the probability of a conflict is p. As mentioned above, the min-conflicts heuristic will not make a mistake if the number of conflicts N_b for each incorrect value is greater than d. We can, therefore, bound the probability of making a mistake by bounding the probability that N_b is less than or equal to d.

To bound N_b, we use Hoeffding's inequality, which states that the sum N of n independent, identically distributed random variables is less than the expected value $\bar{N}$ by more than sn only with probability at most $e^{-2s^2 n}$, for any $s \geq 0$. In our model, N_b is the sum of c potential conflicts, each of which is either 1 or 0, depending on whether there is a conflict. The expected value of N_b is pc. Thus:

$$\Pr(N_b \leq pc - sc) \leq e^{-2s^2 c}$$

Since we are interested in the behavior of the min-conflicts heuristic as d shrinks, let us suppose that d is less than pc. Then, with $s = (pc - d)/c$, we obtain:

$$\Pr(N_b \leq d) \leq e^{-2(pc-d)^2/c}$$

To account for the fact that a mistake can occur if *any* of the $k - 1$ incorrect values has d or fewer conflicts, we bound the probability of making a mistake on any of them by multiplying by $k - 1$:

$$\Pr(mistake) \leq (k-1)e^{-2(pc-d)^2/c}$$

Note that as c (the number of constraints per variable) becomes large, the probability of a mistake approaches zero if all other parameters remain fixed. This analysis thus offers an explanation as to why 3-coloring densely-connected graphs is relatively easy. We also see that as d becomes small, a mistake is also less likely, explaining our empirical observation that having a "good" initial assignment can be important. (Of course, an assignment with few conflicts does not necessarily imply small d, as was illustrated by the 3-colorability problem in figure 6.) In a recent paper, Musick and Russell [35] present an analysis which supports this result. They model heuristic repair algorithms as Markov processes, and show that under this model the choice of initial state has a significant impact on the expected solution time.

Finally, we note that the probability of a mistake also depends on p, the probability that an incorrect value conflicts with another variable's value, and k, the number of values per variable. The probability of a mistake shrinks as p increases or k decreases.

5.2 A Statistical Model for CSP Repair

The simple model presented in the previous section shows, in a qualitative way, how various problem characteristics influence the effectiveness of the min-conflicts heuristic. While the analysis is helpful for understanding how the min-conflicts heuristic works, it is not quantitatively useful, since only very gross characteristics of the problem are considered. In this section we augment the model with statistical assumptions about the task domain, assumptions that enable us to analyze the heuristic's behavior quantititively on particular problems. Specifically, we discard the assumption that there is a uniform probability of a conflict between an erroneous value for a variable and an arbitrary value for any related variable and instead assume that conflicts between variables can be characterized by independent probability distribution functions determined by the problem. We retain the assumption that there is a unique solution. While these assumptions are seldom met in practice on any particular CSP, the augmented model turns out to be a surprisingly accurate predictor of the performance of several heuristics, including min-conflicts, on some interesting classes of problems.

Augmenting the model with statistical assumptions about the task domain provides the basis for a quantitative analysis. The augmented model assumes that conflicts between variables can be characterized by independent probability distribution functions. Also, as in the original model, a single solution is assumed.

While these assumptions may not be precisely met in practice on any particular CSP, the augmented model turns out to be a surprisingly accurate predictor of the performance of several heuristics, including min-conflicts, on some interesting classes of problems.

We continue to assume a binary CSP with n variables and k possible values per variable; for a given assignment, the distance d is the number of variables that must be corrected to obtain a solution. As a measure of heuristic performance, we use the probability that, after a particular repair step, *the distance d is decreased*. This only occurs when the heuristic selects a variable that is assigned an incorrect (non-solution) value and changes it to the unique correct (solution) value. This probability is given by

$$P_{d\to d-1} = P_{\bar{s}} P_{c|\bar{s}},$$

where $P_{\bar{s}}$ is the probability that the variable selection heuristic chooses a variable currently assigned an incorrect (non-solution) value, and $P_{c|\bar{s}}$ is the probability that the value selection heuristic chooses the correct value given that the selected variable has an incorrect value currently assigned. (Subscripts s and $\bar{s}$ indicate variables assigned solution and non-solution values, respectively. For a given variable, the subscripts c and $\bar{c}$ refer to correct and incorrect values, respectively.)

Similarly, the probability of *increasing* the distance from the solution is

$$P_{d\to d+1} = P_s(1 - P_{c|s}),$$

where $P_s = 1 - P_{\bar{s}}$ is the probability that the variable selection heuristic will choose a variable currently assigned a correct value, and $P_{c|s}$ is the probability that the value selection heuristic will choose the correct value given that the chosen variable already has the correct value assigned. The third possibility, that d will remain unchanged, has probability

$$P_{d\to d} = 1 - P_{d\to d-1} - P_{d\to d+1}.$$

The ratio $P_{d\to d-1}/P_{d\to d+1}$ is of particular interest, since as long as it is greater than 1 a heuristic is more likely to move *towards* the solution than *away* from it.

5.3 Conflict Probability Distributions

An expression for the performance measures $P_{d\to d,d\pm 1}$ can be derived for variable and value selection heuristics given the probability distributions for conflicts. Four such distributions are required:

For variables currently assigned the correct value:

$$\theta_{cs}(v) = \begin{bmatrix}\text{Probability that the correct value has } v \text{ conflicts,} \\ 0 \le v \le d\end{bmatrix}$$

$$\theta_{\bar{c}s}(v) = \begin{bmatrix}\text{Probability that an incorrect value has } v \text{ conflicts,} \\ 0 \le v \le n-1\end{bmatrix}$$

For variables currently assigned an incorrect value:

$$\theta_{c\bar{s}}(v) = \begin{bmatrix}\text{Probability that the correct value has } v \text{ conflicts,} \\ 0 \le v \le d-1\end{bmatrix}$$

$$\theta_{\bar{c}\bar{s}}(v) = \begin{bmatrix}\text{Probability that an incorrect value has } v \text{ conflicts,} \\ 0 \le v \le n-1\end{bmatrix}$$

For the cumulative distributions we use the following notation:

$$\theta_{\bar{c}\bar{s}}(> v) = \sum_{w>v} \theta_{\bar{c}\bar{s}}(w).$$

In the remainder of this section we discuss the derivation of these conflict probability distributions θ for two classes of CSPs: those with random independent constraints, and those with more structured constraints. For the readers convenience. Table 4 summarizes the notation we employ.

n Number of variables
k Values per variable
c Binary constraints per variable
d Distance to solution (number of variables with incorrect values)
$P_{d \to d-1}$ Probability that after a repair step d decreases.
$P_{d \to d+1}$ Probability that after a repair step d increases.
$P_{d \to d}$ Probability that after a repair step d is unchanged.

In general, subscripts s and $\bar{s}$ indicate variables assigned solution and non-solution values, respectively. For a given variable, the subscripts c and $\bar{c}$ refer to correct and incorrect values, respectively. For example:

$P_{\bar{s}}$ Probability that the variable choosen is currently assigned a non-solution (i.e., incorrect) value.
$P_{c|\bar{s}}$ Probability of choosing a correct value, given that a non-solution value is currently assigned.
$\theta_{\bar{c}s}(v)$ For a variable currently assigned a solution value, probability that an incorrect value has v conflicts.
$p_{c \Rightarrow \bar{c}}$ Probability that a correct value for variable V conflicts with an incorrect value for variable V'

Table 4: Summary of Notation

5.3.1 Random CSPs

Random CSPs can be characterized by two probabilities as follows:

- $p_{c \Rightarrow \bar{c}} \equiv p_{\bar{c} \Rightarrow c}$ is the probability that a *correct* value for variable V conflicts with an *incorrect* value for variable V', and
- $p_{\bar{c} \Rightarrow \bar{c}}$ is the probability that an *incorrect* value for variable V conflicts with an *incorrect* value for variable V',

Note that, by definition, $p_{c \Rightarrow c} = 0$ (there can be no conflicts between correct values).

Consider a state in which there are d variables assigned incorrect values. If a variable is assigned the correct value, then it can conflict with at most the d variables assigned incorrect values. Assuming that *the probability of each conflict is independent*, the total number of conflicts follows a binomial distribution:

$$B(x, p, N) = \binom{N}{x} p^x (1-p)^{N-x}$$

where x is the number of "successes", p is the probability of success in a single "trial", and N is the number of trials. Thus

$$\theta_{cs}(v) = B(v, p_{\bar{c} \Rightarrow c}, d).$$

Incorrect values can conflict with the d incorrectly assigned variables, each with probability $p_{\bar{c} \Rightarrow \bar{c}}$, and with the other $n - d - 1$ correctly assigned variables, each with probability $p_{c \Rightarrow \bar{c}}$. The distribution is:

$$\theta_{\bar{c}s}(v) = \sum_{k=0}^{v} B(k, p_{\bar{c} \Rightarrow \bar{c}}, d) B(v-k, p_{c \Rightarrow \bar{c}}, n-d-1).$$

This is the distribution for the sum of two binomially-distributed variables with different values for N and p. In the case where $p_{\bar{c} \Rightarrow \bar{c}} = p_{c \Rightarrow \bar{c}} = p_c$, this reduces to $\theta_{\bar{c}s}(v) = B(v, p_c, n-1)$.

For variables currently assigned incorrect values, the correct value can conflict with at most the $d - 1$ other variables assigned incorrect values, each with probability $p_{c \Rightarrow \bar{c}}$:

$$\theta_{c\bar{s}}(v) = B(v, p_{c \Rightarrow \bar{c}}, d-1).$$

Problem	c	k'	k	$p_{c\Rightarrow\bar{c}}$	$p_{\bar{c}\Rightarrow\bar{c}}$
Sparse graph 3-colorability VD=4	4	1	3	$\frac{2}{n-1}$	$\frac{1}{n-1}$
Dense graph 3-colorability VD=$2n/3$	$\frac{2n}{3}$	1	3	$\frac{1}{3}\frac{n}{n-1}$	$\frac{1}{6}\frac{n}{n-1}$
Dechter-Pearl general case	$p_1 n$	$(1-p_2)k$	k	$\frac{p_1(1-p_2)kn}{(k-1)(n-1)}$	$\frac{p_1(1-p_2)k(k-2)n}{(k-1)^2(n-1)}$
Dechter-Pearl $k=5$ $p_1=0.5, p_2=0.6$	$\frac{n}{2}$	2	5	$\frac{1}{4}\frac{n}{n-1}$	$\frac{3}{16}\frac{n}{n-1}$

Table 5: Probabilities of conflicts between solution and non-solution values $p_{c\Rightarrow\bar{c}}$, and between non-solution and non-solution values $p_{\bar{c}\Rightarrow\bar{c}}$, for some CSPs that can be treated as "random". For graph 3-colorability problems the mean vertex degree (VD) of the problem graph is indicated. The Dechter-Pearl problem, shown for comparison, has probability p_1 of a constraint between variables, and p_2 that a constraint permits any specific pair of values. c is the mean number of variables constrained by any variable, k' is the mean number of values prohibited by a constraint between two variables and k is the domain size.

Incorrect values can conflict with the other $d-1$ incorrect variables, each with probability $p_{\bar{c}\Rightarrow\bar{c}}$, and by the $n-d$ correct variables, each with probability $p_{c\Rightarrow\bar{c}}$. The distribution function is:

$$\theta_{\bar{c}\bar{s}}(v) = \sum_{k=0}^{v} B(k, p_{\bar{c}\Rightarrow\bar{c}}, d-1) B(v-k, p_{c\Rightarrow\bar{c}}, n-d).$$

In the case where $p_{\bar{c}\Rightarrow\bar{c}} = p_{c\Rightarrow\bar{c}} = p_c$, this reduces to $\theta_{\bar{c}\bar{s}}(v) = B(v, p_c, n-1) = \theta_{\bar{c}s}(v)$.

To calculate $p_{\bar{c}\Rightarrow\bar{c}}$ and $p_{c\Rightarrow\bar{c}}$ suppose that each variable constrains on average c other variables, and, if there is a constraint between any two variables V and V', then each value for V conflicts with an average k' values for V'. Then the probability that V constrains V' is $c/(n-1)$, and the probability that the correct value for V conflicts with an incorrect value for V' is $k'/(k-1)$, where k is the domain size. Thus we have

$$p_{c\Rightarrow\bar{c}} = \frac{c}{n-1}\frac{k'}{k-1}.$$

A similar argument for incorrect values yields

$$p_{\bar{c}\Rightarrow\bar{c}} = \frac{c}{n-1}\frac{k'}{k-1}\frac{k-2}{k-1} = \frac{k-2}{k-1} p_{c\Rightarrow\bar{c}}.$$

Values for $p_{c\Rightarrow\bar{c}}$ and $p_{\bar{c}\Rightarrow\bar{c}}$ are given in Table 5 for some illustrative problem types, including sparse and dense graph 3-colorability problems. For comparison, the table also shows the corresponding values for the random problem described by Dechter and Pearl [8].

5.3.2 Highly-Structured CSPs

The conflict distribution functions for random CSPs derived above predict significant variance in conflict counts in the solution state. For example, when $d=0$ the distribution $\theta_{\bar{c}s}(v)$ reduces to $B(v, p_{c\Rightarrow\bar{c}}, n-1)$

which has mean $(n-1)p_{c \Rightarrow \bar{c}}$ and variance $(n-1)p_{c \Rightarrow \bar{c}}(1-p_{c \Rightarrow \bar{c}})$. For some CSPs, the variance in the solution state is demonstrably much less than this, and can be essentially zero for problems with sufficiently strong regularities. For example, treating n-queens as random would predict that many incorrect values would have zero conflicts for large n, but in fact, in the solution state, each incorrect value has at least one conflict. This structure can be incorporated into the calculation of θ, as illustrated in Appendix A for a simplified n-queens model which assumes that exactly three other queens conflict with each incorrect value.

5.4 Value Selection Heuristics

In this section we derive expressions for the probability of choosing a correct value ($P_{c|s}$ and $P_{c|\bar{s}}$) based on the conflict probability distributions defined in Section 5.3. It is important to note that the derived probabilities depend only on the existence of the θ distributions, and not on their specific form.

5.4.1 Min-Conflicts Value Selection

The min-conflicts value selection heuristic can be stated as:

> Choose a value which has the *minimum* number of conflicts with the assigned values for the other variables. If there is more than one such value, select one at random.

Note that with this rule there need be no change in the assignment.

$P_{c|s}$: variable with correct value assigned

Conflicts on the correct value must be due to one or more of the d variables which have incorrect assignments. Suppose there are $v > 0$ conflicts on the correct value (if there are $v = 0$ conflicts, the variable would not have been selected for repair). We seek the probability of leaving the assigned value unchanged, which is the right decision in this case. If any of the $k-1$ incorrect values has less than v conflicts, then the min-conflicts heuristic will choose one of these values. The correct value will be chosen only if all $k-1$ incorrect values have at least v conflicts. Of the $k-1$ incorrect values, let m be the number which have exactly v conflicts, while the remaining $k-1-m$ have $> v$ conflicts. The probability of such a configuration is:

$$\theta_{\bar{c}s}(v)^m \theta_{\bar{c}s}(> v)^{k-1-m}$$

while the total number of such configurations is $\binom{k-1}{m}$. Since, in this configuration, there are m values other than the correct value with an equal number v of conflicts, the probability of choosing the correct value is $1/(m+1)$. Thus the total probability of choosing the correct value, given that it has v conflicts, is:

$$P^{\text{sol}}(v) = \sum_{m=0}^{k-1} \binom{k-1}{m} \theta_{\bar{c}s}(v)^m \theta_{\bar{c}s}(> v)^{k-1-m} \frac{1}{m+1}.$$

The probability of v conflicts on the correct value, given that it has > 0 conflicts, is $\theta_{cs}(v)/[1-\theta_{cs}(0)]$. Combining these yields the total probability that the heuristic will leave the assignment unchanged:

$$P_{c|s} = \sum_{v=1}^{d} \frac{\theta_{cs}(v)}{1-\theta_{cs}(0)} P^{\text{sol}}(v).$$

$P_{c|\bar{s}}$: variable with incorrect value assigned

Suppose the number of conflicts on the correct value is v, and that there are w conflicts on the current (incorrect) assigned value. Let $P^{\text{sol}}(v,w)$ denote the probability of choosing the correct value in this situation. There are three cases:

(1) $v > w$: The correct value will not be chosen since the current value has fewer conflicts, so $P^{\mathrm{sol}}(v,w)\big|_{v>w} = 0$.

(2) $v = w$: In this case we have to consider the other $k-2$ incorrect values. Summing over configurations where m have exactly v conflicts, and the remaining $k-2-m$ have $> v$ conflicts, yields:

$$P^{\mathrm{sol}}(v,w)\big|_{v=w} \equiv R^{v=w}(v) = \sum_{m=0}^{k-2} \binom{k-2}{m} \theta_{\bar{c}\bar{s}}(v)^m \theta_{\bar{c}\bar{s}}(>v)^{k-2-m} \frac{1}{m+2}$$

(3) $v < w$: Similar to case (2) except that in this case the heuristic will certainly not leave the assignment unchanged, so the probability of choosing the correct value increases from $1/(m+2)$ to $1/(m+1)$:

$$P^{\mathrm{sol}}(v,w)\big|_{v<w} \equiv R^{v<w}(v) = \sum_{m=0}^{k-2} \binom{k-2}{m} \theta_{\bar{c}\bar{s}}(v)^m \theta_{\bar{c}\bar{s}}(>v)^{k-2-m} \frac{1}{m+1}$$

The total probability of choosing the correct value is

$$P_{c|\bar{s}} = \sum_{w=1}^{n-1} \sum_{v=0}^{d-1} \theta_{c\bar{s}}(v) \frac{\theta_{\bar{c}\bar{s}}(w)}{1-\theta_{\bar{c}\bar{s}}(0)} P^{\mathrm{sol}}(v,w),$$

using the fact that the probability of v conflicts on an incorrect value, given that the value has > 0 conflicts, is $\theta_{\bar{c}\bar{s}}(v)/\left[1-\theta_{\bar{c}\bar{s}}(0)\right]$.

5.4.2 Random-Conflicts Value Selection

The min-conflicts heuristic examines the *number* of conflicts on each value to determine which to assign. A less-informed heuristic could simply check *whether or not* there are any conflicts on values. This approach is captured by the "random-conflicts" rule:

> If one or more values has *no* conflicts, select one of these values (at random). If *all* values have conflicts, select one at random.

The assignment is not required to change (although it must change if at least one value has zero conflicts).

The derivation of $P_{c|s}$ and $P_{c|\bar{s}}$ follows the same argument as above, with the results:

$$P_{c|s} = \theta_{cs}(>0)^{k-1}\frac{1}{k}.$$

and

$$P_{c|\bar{s}} = \theta_{c\bar{s}}(0)\ P^{\mathrm{sol}}(v,w)\big|_{v=0} + \left[1-\theta_{c\bar{s}}(0)\right]\ P^{\mathrm{sol}}(v,w)\big|_{v>0},$$

where

$$P^{\mathrm{sol}}(v,w)\big|_{v=0} = \sum_{m=0}^{k-2} \binom{k-2}{m} \theta_{\bar{c}\bar{s}}(0)^m \theta_{\bar{c}\bar{s}}(>0)^{k-2-m} \frac{1}{m+1},$$

$$P^{\mathrm{sol}}(v,w)\big|_{v>0} = \theta_{\bar{c}\bar{s}}(>0)^{k-2}\frac{1}{k}.$$

$P^{\mathrm{sol}}(v,w)$ is the probability of choosing the correct value for a variable with v conflicts on the correct value and $w > 0$ conflicts on an incorrect value.

5.4.3 Random Value Selection

This is the "least-possible-informed" value selection rule:

> Select a value at random, regardless of conflicts.

With this rule, the probability of choosing the correct value is independent of the variable's currently assigned value:

$$P_{c|s} = P_{c|\bar{s}} = 1/k$$

5.5 Variable Selection

In this section we develop expressions for the probability of selecting a variable to be repaired (P_s or $P_{\bar{s}}$) based on the following simple rule:

Select for repair a variable at random from the set of all variables that are currently in conflict.

Consider first a variable that is assigned an incorrect value. The probability that there are one or more conflicts on its assigned value is $1 - \theta_{\bar{c}\bar{s}}(0)$. Since there are a total of d such variables, the expected number with conflicts is

$$N_{\bar{s},\mathrm{conf}} = d\,[1 - \theta_{\bar{c}\bar{s}}(0)]\,.$$

Now consider a variable that is assigned a correct value. The probability that there are one or more conflicts on its assigned value is $1 - \theta_{cs}(0)$. Since there are a total of $n - d$ such variables, the expected number with conflicts is

$$N_{s,\mathrm{conf}} = (n - d)\,[1 - \theta_{cs}(0)]\,.$$

Thus, for a variable with conflicts that is picked at random, the probability that it is currently assigned a correct value is:

$$P_s = \frac{N_{s,\mathrm{conf}}}{N_{\bar{s},\mathrm{conf}} + N_{s,\mathrm{conf}}},$$

while the probability that it is currently assigned an incorrect value is:

$$P_{\bar{s}} = 1 - P_s = \frac{N_{\bar{s},\mathrm{conf}}}{N_{\bar{s},\mathrm{conf}} + N_{s,\mathrm{conf}}}.$$

5.6 Evaluation of the Statistical Model

We have numerically evaluated the expressions above for $P_{d\to d,d\pm1}$, $P_{c|\bar{s}}$, $P_{c|s}$, etc. on two random CSP problem types, and on the simplified n-queens model, in order to compare the predicted performance of the three value selection heuristics discussed above. For the random CSPs we have also generated sample problems and computed the probabilities empirically for comparison with the model. These results are described in this section.

5.6.1 Random CSPs

We have taken two graph 3-colorability problems for comparison of the heuristics:

- **H3C:** "Hard" 3-colorability, random sparsely-connected graph, mean vertex degree = 4. In the solution state the expected number of conflicts on incorrect values is 2, approximately independent of problem size n.
- **E3C:** "Easy" 3-colorability, random densely-connected graph, mean vertex degree = $2n/3$. In the solution state the expected number of conflicts on incorrect values is $n/3$, i.e. increasing linearly with problem size

The relevant conflict probabilities for these two problems are given in Table 5. Probabilities were calculated for both problem types for $n = 90$. Value selection heuristics are labelled as follows in the figures: **MC** min-conflicts (Section 5.4.1); **RC** random-conflicts (Section 5.4.2); and **R** random (Section 5.4.3).

Variable selection

Fig. 7 shows $P_{\bar{s}}$ vs. d/n, the probability that a variable currently assigned an incorrect value will be chosen for repair. The probability is lower for the densely-connected E3C problem, since even a small number of incorrectly assigned variables can introduce a large number of conflicts.

Figure 7: Probability of selecting a variable that is assigned an incorrect value for H3C and E3C random problems.

Figure 8: Probability of choosing correct values for variables currently assigned correct or incorrect values.

Value selection

Fig. 8 compares value selection for the two problems. Here it is desirable that both $P_{c|s}$ (Fig. 8a,b) and $P_{c|\bar{s}}$ (Fig. 8c,d) be as large as possible. Random value selection (labelled R in the figures) has uniform probability 1/3 of making the correct choice in both problems. For H3C variables with correct values assigned (Fig. 8a), RC does worse than random, and MC does better only for small d/n. In contrast, for variables that have incorrect values (Fig. 8c), the probability is fairly high for both MC and RC that the correct value will be selected, with MC showing slightly better performance. For E3C (Fig. 8b,d), MC has probability near unity of choosing the correct value, whether or not the current value is correct. RC does no better than random except for variables currently assigned incorrect values and $d/n < 0.2$ (Fig. 8d).

Combined Variable and Value selection

Fig. 9 shows the probabilities of moving towards ($P_{d\rightarrow d-1}$, Fig. 9a,b) or away from ($P_{d\rightarrow d+1}$, Fig. 9c,d) the solution for the variable selection method combined with each of the three value selection methods. For H3C (Fig. 9a,c), all three value selection methods have higher probability of *worsening* the state than of improving it. MC shows the best performance, with the largest values for $P_{d\rightarrow d-1}$ and the smallest for $P_{d\rightarrow d+1}$ in the range $d/n < 2/3$. For E3C (Fig. 9b,d), both RC and R tend to worsen the state, while MC has a much higher probability of improving it.

The ratio $P_{d\rightarrow d-1}/P_{d\rightarrow d+1}$ provides a useful comparison of combined variable and value selection performance: it is greater than unity when a heuristic is more likely to improve the state than to worsen it. Fig. 10 plots this ratio on a logarithmic scale vs. d/n for each of the three value selection methods. For H3C (Fig. 10a), MC is best (for $d \ll n$), followed by RC and R, but in all cases the ratio is < 1. For E3C (Fig. 10b) the results are very different: MC shows a much higher chance of improving the state, while both RC and R worsen it. RC is significantly better than R only for very small d/n.

Comparison with Empirical Results

To see how well the model captures features of the heuristics when applied to actual problems, we have generated random problem instances with known solutions[6], then assigned incorrect values to some of the variables and calculated empirically the same probabilities that are predicted by the statistical model. Fig. 11 shows the comparison for MC value selection: the empirical data points, indicated by the $+$ and $\times$ symbols, show the results of averaging 200 states for each value of d. The agreement with the model probability calculations is excellent.

5.6.2 N-Queens

We have evaluated the simplified n-queens model of Section 5.3.2 and Appendix A for min-conflicts value selection. Fig. 12 shows the quantities $P_{d\rightarrow d-1}$, $P_{d\rightarrow d+1}$, and the ratio $P_{d\rightarrow d-1}/P_{d\rightarrow d+1}$ for small d for n=64, 96, 128, and 256. As n increases, the relative probability of moving towards the solution increases as well. While this is in accord with the experimental results, the model does not permit more quantitative comparison due to the simplifying assumption that the mean conflicts on incorrect values is 3 (instead of the actual ~ 2.5). The situation for n-queens is further complicated by the fact that solutions appear to be relatively numerous, violating the model assumption that there is a unique solution.

5.7 Limiting Behavior for Random CSPs

There are two interesting limiting cases of the model for random CSPs, corresponding to limiting forms of the conflict probability distribution functions θ (see Section 5.3.1). These limits are discussed in this section.

[6] The random problem instances were not guaranteed to have unique solutions; simple relabelling of colors will yield several.

Figure 9: Probability of moving towards ($P_{d \to d-1}$) or away from ($P_{d \to d+1}$) the solution.

Figure 10: $P_{d \to d-1}/P_{d \to d+1}$ for the three value selection heuristics.

Figure 11: Comparison of predicted results with empirical results for min-conflicts value selection.

Figure 12: Performance probabilities for the n-queens model.

Figure 13: Scaling behavior with n for variable selection method.

5.7.1 Poisson Limit

In the case $n \to \infty$, $p_{\bar{c} \Rightarrow \bar{c}} \to p_{c \Rightarrow \bar{c}} = p_c$, and $np_c \to$ constant, the conflict distribution functions approach the Poisson distribution: $\theta_{cs}(v) \approx \theta_{c\bar{s}}(v) \approx P_{\text{poisson}}(v, dp_c)$, and $\theta_{\bar{c}s}(v) \approx \theta_{\bar{c}\bar{s}}(v) \approx P_{\text{poisson}}(v, np_c)$, where $P_{\text{poisson}}(v, \mu) = e^{-\mu}\mu^v/v!$. If we let $d = fn$, i.e. f is the fraction of variables assigned incorrect values, we can write the distributions for $\theta_{cs}(v)$ and $\theta_{c\bar{s}}(v)$ as:

$$\theta_{cs}(v) \approx \theta_{c\bar{s}}(v) \approx \frac{(e^{-\mu})^f (f\mu)^v}{v!},$$

where $\mu = np_c$. The result is independent of n, and thus we have the important conclusion that *the performance of value selection heuristics depends only on d/n in the Poisson limit $p_c \propto 1/n$ for small np_c.* This is also true of the variable selection method used in the model (which depends only on $\theta_{\bar{c}\bar{s}}(0)$ and $\theta_{cs}(0)$). Fig. 13a illustrates this dependence on d/n for the H3C problem for n=30, 60, and 90: the differences are already nearly indistinguishable.

5.7.2 Gaussian Limit

At the other extreme, consider the case when the mean number of conflicts *increases* with n, e.g. when $p_{c \Rightarrow \bar{c}}$ is approximately constant, and $np_{c \Rightarrow \bar{c}}$, the expected number of conflicts for an incorrect value for a variable when in the solution state, increases linearly with n. In this case, for sufficiently large n, the distributions can be approximated by Gaussian distributions with mean $np_{c \Rightarrow \bar{c}}$ and variance $\sigma^2 = np_{c \Rightarrow \bar{c}}(1 - p_{c \Rightarrow \bar{c}})$. We can derive the dominant behavior of min-conflicts value selection in the limit $n \gg d \gg 1$ by approximating the sums in the expressions for $P_{c|\bar{s}}$ and $P_{c|s}$ by integrals over the Gaussian distribution. Only values near the peak of the Gaussian make significant contributions, and in the limit $P_{c|\bar{s}} \approx P_{c|s} \approx 1$. The probability of choosing a variable with an incorrect value becomes $P_{\bar{s}} \approx d/n$ since $N_{\bar{s},\text{conf}} \approx d$ and $N_{s,\text{conf}} \approx n - d$. From this it follows that $P_{d \to d-1} \approx d/n$ and $P_{d \to d+1} \approx 0$. This linear dependence of $P_{d \to d-1}$ on d for large n is evident in Fig. 13b, which shows $P_{d \to d-1}$ and $P_{d \to d+1}$ for n=30, 60, and 90 for MC value selection.

5.7.3 Global Performance of Min-conflicts Hill-climbing Repair

The simple limiting forms above permit some general statements to be made about the behavior of hill-climbing repair methods based on min-conflicts value selection. Hill-climbing repair can be modelled as a random (Markovian) walk described by the probabilities $P_{d \to d, d\pm 1}$ of moving towards or away from an "absorbing barrier" at $d = 0$.

In the Gaussian limit where $P_{d \to d+1} \approx 0$, $P_{d \to d-1} \approx d/n$ (cf. Fig. 9b), the expected number of hill-climbing steps to transition from d to $d - 1$ is $1/P_{d \to d-1} = n/d$. From an initial distance d_0, the expected number of steps t to reach $d = 0$ is thus

$$t_{d_0 \to 0} = \sum_{i=1}^{d_0} \frac{n}{i} \approx n \left[\gamma + \ln d_0 + O(\frac{1}{d_0^2}) \right]$$

where $\gamma = 0.577 \cdots$ is Euler's constant. Thus *the expected number of steps to reach the solution is linear in the problem size n and depends only logarithmically on how far away the initial guess is from the solution.*

In the Poisson limit where $P_{d \to d+1} > P_{d \to d-1}$ but both are nearly constant (cf. Fig. 9a), the distance from the solution after t steps can be written as $d(t) = d_0 + \sum_{i=1}^{t} \xi_i$ where ξ_i is a random variable representing the change in d with each step. The probability distribution for ξ has mean $\mu = P_{d \to d+1} - P_{d \to d-1}$ and variance $\sigma^2 = P_{d \to d+1} + P_{d \to d-1} - (P_{d \to d+1} - P_{d \to d-1})^2$. After a sufficiently large number of steps, the distribution for $d(t)$ is approximately Gaussian with mean $\mu_d = d_0 + t\mu$ and variance $\sigma_d^2 = t\sigma^2$. The mean μ_d represents a drift of the expected value of $d(t)$ *away* from the solution $d = 0$. The probability of reaching the solution after t steps is approximately given by the tail of the Gaussian distribution for $d \leq 0$, which approaches

$$\frac{\sigma}{\mu\sqrt{2\pi t}} \exp\left(-\frac{\mu^2 t}{2\sigma^2}\right)$$

for large t. The important point is the predicted *exponential decline in the probability of reaching the solution as the number of hill-climbing steps increases.* This result provides an explanation for the observed behavior of the GDS network and of min-conflicts hill climbing on sparse 3-colorable graphs as described above in Section 4.3: when the the number of steps is limited to $t \propto n$, there is an exponential decline with problem size n of the probability of finding the solution.

5.8 Summary and Caveats

The statistical model of CSP repair described here is a surprisingly good predictor of "conflict-informed" value selection performance for random CSPs. The model has both theoretical and practical benefits. It permits average-case comparisons of different variable and value selection heuristics, from which can be drawn general conclusions about their relative effectiveness. For particular problem types, limiting behavior for large n can be derived, including general statements as to whether heuristics will show better or worse performance as problem size increases. For random CSPs discussed in detail above, these conclusions include:

- min-conflicts is the most effective value selection method among those considered;
- min-conflicts performs relatively better as n increases, particularly when $p_{c \Rightarrow \bar{c}}$ increases with n or remains constant;
- if the Gaussian limit applies, then hill climbing with min-conflicts is an effective repair strategy, showing only weak dependence on the initial guess and $O(n)$ dependence on problem size n;
- if the Poisson limit applies, then the probability of reaching the solution declines exponentially with the number of hill-climbing steps.

Application of the model to other problem types is the subject of future research.

There are, however, several factors that limit the applicability of the model. The most important are that conflicts are assumed to be independent, and that a single solution state is assumed. The presence of multiple solutions may not be a serious limitation so long as the model is applied in the vicinity of a solution, and that solutions are not so dense as to render this meaningless. Conflict independence is more significant, since highly structured problems which occur in practice may violate this assumption. Nevertheless, to the extent that the statistical properties of classes of problems can be established, it may still be possible to use the model to perform average-case analysis of heuristics.

Two other limitations are worth noting, since we have analyzed the min-conflicts heuristic independent of the initialization process and search strategy. First, the model permits no conclusions about the assignment being repaired, yet the construction of a good initial guess (i.e. an assignment such that d is small) is a key problem for repair methods. Second, since the model ignores all fine structure in the problem, the possibility of pathological configurations is not considered. This can manifest itself in hill-climbing techniques as "cycles", where the same variables are repaired again and again, but no progress is made towards the solution. To model the performance of the min-conflicts heuristic in conjunction with a particular search strategy, such as hill-climbing, a more detailed analysis is required. For example, in a recent paper, Morris [33] examines the structure of the n-queens problem, and shows analytically that, for min-conflicts hill-climbing, almost all local minima are solutions.

6 Discussion

The heuristic hill-climbing method described in this paper can be characterized as a *local search* method[20], in that each repair minimizes the number of conflicts for an individual variable. Local search methods have been applied to a variety of important problems, often with impressive results. For example, the Kernighan-Lin method, perhaps the most successful algorithm for solving graph-partitioning problems, repeatedly improves a partitioning by swapping the two vertices that yield the greatest cost differential. The much-publicized simulated annealing method can also be characterized as a form of local search[19]. However, it is well-known that the effectiveness of local search methods depends greatly on the particular task.

In fact, it is easy to imagine problems on which the min-conflicts heuristic will fail. The heuristic is poorly suited to problems with a few highly critical constraints and a large number of less important constraints. For example, consider the problem of constructing a four-year course schedule for a university student. We may have an initial schedule which satisfies almost all of the constraints, except that a course scheduled for the first year is not actually offered that year. If this course is a prerequisite for subsequent courses, then many significant changes to the schedule may be required before it is fixed. In general, if repairing a constraint violation requires completely revising the current assignment, then the min-conflicts heuristic will offer little guidance. This intuition is partially captured by the analysis presented in the previous section, which shows that the effectiveness of the heuristic is inversely related to the distance to a solution.

The problems investigated in this paper, especially the HST and n-queens problem, tend to be relatively uniform in that critical constraints rarely occur. In part, this is due to the way the problems are represented. For example, in the HST problem, as described earlier, the transitive closure of temporal constraints is explicitly represented. A single "after" relation, for example, can thus be transformed into a set of "after" relations. This improves performance because the min-conflicts heuristic is less likely to violate a set of constraints than a single constraint. In some cases, we expect that more sophisticated techniques will be necessary to identify critical constraints[11]. To this end, we are currently evaluating explanation-based learning techniques [9] as a method for identifying critical constraints.

The algorithms described in this paper also have an important relation to previous work in AI. In particular, there is a long history of AI programs that use repair or debugging strategies to solve problems, primarily in the areas of planning and design[37, 40]. This approach has recently had a renaissance with the emergence of case-based[14, 26] and analogical [17, 24, 42] problem solving. To solve a problem, a case-based system will retreive the solution from a previous, similar problem and repair the old solution so that it solves the new problem.

The fact that the min-conflicts approach performs well on n-queens, a well-studied, "standard" constraint-satisfaction problem, suggests that AI repair-based approaches may be more generally useful than previously thought. Additional evidence also comes from a very recent study by Selman, Levesque and Mitchell [36], in which they showed that a repair-based algorithm (very similar to the hill-climbing algorithms investigated here) performs well on hard satisfiability problems. However, as we have pointed out, in some cases it can be more time-consuming to repair a solution than to construct a new one from scratch. It may be that our analysis of min-conflicts for CSP problems can be extended to repair methods for other tasks, such as case-based planning methods. We conjecture that for each of the factors affecting the performance of min-conflicts, such as the expected "distance" from the initial assignment to the solution and the degree that each variable is constrained, there are analogous factors for other tasks.

There are many possible extensions to the work reported here, but three are particularly worth mentioning. First, we expect that there are other applications for which the min-conflicts approach will prove useful. Conjunctive matching, for example, is an area where preliminary results appear promising. This is particularly true for matching problems that require only that a good partial-match be computed. Second, we expect that there are interesting ways in which the min-conflicts heuristic could be combined with other heuristics. For example, as mentioned earlier, when a "most-conflicted" variable ordering strategy is used together with min-conflicts, the resulting program outperforms min-conflicts alone on graph 3-colorability problems. Finally, there is the possibility of employing the min-conflicts heuristics with other search techniques. In this paper, we only considered two very basic methods, hill climbing and backtracking. However, more sophisticated techniques such as best-first search are obvious candidates for investigation, since the number of conflicts in an assignment can serve as a heuristic evaluation function. Another possibility is Tabu search[16], a hill-climbing technique that maintains a list of forbidden moves in order to avoid cycles. Morris[31, 32] has also proposed a hill-climbing method which can break out of local maxima by systematically altering the cost function. The work by Morris and much of the work on Tabu search bears a close relation to our approach.

7 Conclusions

In this paper we have analyzed a very successful neural network algorithm and shown that a simple heuristic search method behaves similarly. Specifically, we carried out extensive experiments in three task domains in which the min-conflicts hill-climbing algorithm and the GDS network exhibited similar performance. Based on our experience with both programs, we conclude that the min-conflicts heuristic captures the critical aspects of the GDS network. In this sense, we have explained why the network is so effective.

We have also demonstrated that the min-conflicts heuristic can be employed in conjunction with other types of symbolic search methods besides hill-climbing. In particular, we showed that it can be used as a value-ordering heuristic by an informed backtracking algorithm. This is an important consideration, since we expect that in many applications the choice of search strategy may be critical to producing satisfactory solutions.

By isolating the min-conflicts heuristic from the search strategy, we distinguished the idea of a repair-based CSP method from the particular strategy employed to search within the space of repairs. This enabled us to carry out a strategy-independent analysis of the heuristic. The analysis identified several factors that effected the utility of the min-conflicts heuristic, such as the expected distance between the initial assignment and the solution. We believe that this analysis may be relevant to repair-based problem solving methods in general.

There are also several practical implications of this work. First, the scheduling system for the Hubble Space Telescope, SPIKE, now employs our symbolic method, rather than the network, reducing the overhead necessary to arrive at a schedule. Perhaps even more importantly, it is easy to experiment with variations of the symbolic method, which should facilitate transferring SPIKE to other scheduling applications. Finally, by demonstrating that repair-based methods are applicable to standard constraint satisfaction problems, such as N-queens, we have provided a new tool for solving CSP problems.

8 Acknowledgements

The authors wish to thank Hans-Martin Adorf, Don Rosenthal, Richard Franier, Peter Cheeseman and Monte Zweben for their assistance and advice. We also thank Ron Musick and our anonymous reviewers for their comments. The Space Telescope Science Institute is operated by the Association of Universities for Research in Astronomy for NASA.

References

[1] B. Abramson and M. Yung. Divide and conquer under global constraints: A solution to the n-queens problem. *Journal of Parallel and Distributed Computing*, 61:649–662, 1989.

[2] H.M. Adorf and M.D. Johnston. A discrete stochastic neural network algorithm for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Neural Networks*, San Diego, CA, 1990.

[3] E. Biefeld and L. Cooper. Bottleneck identification using process chronologies. In *Proceedings IJCAI-91*, Sydney, Australia, 1991.

[4] J. Bitner and E.M. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18:651–655, 1975.

[5] G. Brassard and P. Bratley. *Algorithmics - Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[6] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[7] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the *really* hard problems are. In *Proceedings IJCAI-91*, Sydney, Australia, 1991.

[8] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[9] M. Eskey and M. Zweben. Learning search control for constraint-based scheduling. In *Proceedings AAAI-90*, Boston, Mass, 1990.

[10] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Morgan Kaufmann Publishers, Inc., 1987.

[11] M.S. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings IJCAI-89*, Detroit, MI, 1989.

[12] E.C. Freuder. Partial constraint satisfaction. In *Proceedings IJCAI-89*, Detroit, MI, 1989.

[13] M.L. Ginsberg and W.D. Harvey. Iterative broadening. In *AAAI Proceedings*, 1990.

[14] K.J. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory.* PhD thesis, Yale University, Department of Computer Science, 1986.

[15] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[16] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[17] A.K. Hickman and M.C. Lovett. Partial match and search control via internal analogy. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, Ill., 1991.

[18] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume 79, 1982.

[19] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, Part II. *Journal of Operations Research*, 1990.

[20] D.S. Johnson, C.H. Papadimitrou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[21] M.D. Johnston. Automated telescope scheduling. In *Proceedings of the Symposium on Coordination of Observational Projects.* Cambridge University Press, 1987.

[22] M.D. Johnston and H.M. Adorf. Learning in stochastic neural networks for constraint satisfaction problems. In *Proceedings of NASA Conference on Space Telerobotics*, Pasadena, CA, January 1989.

[23] L.V. Kale. An almost perfect heuristic for the n nonattacking queens problem. *Information Processing Letters*, 34:173–178, 1990.

[24] S. Kambhampati. Supporting flexible plan reuse. In Minton S., editor, *Machine Learning Methods for Planning and Scheduling.* Morgan Kaufmann, 1992.

[25] N. Keng and D.Y.Y. Yun. A planning/scheduling methodology for the constrained resource problem. In *Proceedings IJCAI-89*, Detroit, MI, 1989.

[26] J.L. Kolodner, R.L.Jr. Simpson, and K. Sycara-Cyranski. A process model of case-based reasoning in problem solving. In *Proceedings IJCAI-85*, Los Angeles, CA, 1985.

[27] C.R. Kurtzman. *Time and Resource Constrained Scheduling, with Applications to Space Station Planning*. PhD thesis, Dept. of Aeronautics and Astronautics, MIT, Cambridge, MA, 1988.

[28] C.R. Kurtzman and D.L. Aiken. The Mfive space station crew activity scheduler and stowage logistics clerk. In *Proceedings the AIAA Computers in Aerospace VII Conference*, Monterey, CA, 1989.

[29] P. Langley. Systematic and nonsystematic search strategies. In *Proceedings AAAI-92*, San Jose, CA, 1992.

[30] S. Minton, M. Johnston, A.B. Philips, and P. Laird. Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings AAAI-90*, 1990.

[31] P. Morris. Solutions without exhaustive search: An iterative descent method for binary constraint satisfaction problems. In *Proceedings the AAAI-90 Workshop on Constraint-Directed Reasoning*, Boston, MA, 1990.

[32] P. Morris. An iterative improvement algorithm with guaranteed convergence. Technical Report TR-M-91-1, Intellicorp Technical Note, 1991.

[33] P. Morris. On the density of solutions in equilibrium points for the queens problem. In *Proceedings AAAI-92*, San Jose, CA, 1992.

[34] N. Muscettola, S.F. Smith, G. Amiri, and D. Pathak. Generating space telescope observation schedules. Technical Report CMU-RI-TR-89-28, Carnegie Mellon University, Robotics Institute, 1989.

[35] R. Musick and S. Russell. How long will it take? In *Proceedings AAAI-92*, San Jose, CA, 1992.

[36] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings AAAI-92*, San Jose, CA, 1992.

[37] R.G. Simmons. A theory of debugging plans and interpretations. In *Proceedings AAAI-88*, Minneapolis, MN, 1988.

[38] R. Sosic and J. Gu. A polynomial time algorithm for the n-queens problem. *SIGART*, 1(3), 1990.

[39] H.S. Stone and J.M. Stone. Efficient search techniques - an empirical study of the n-queens problem. *IBM Journal of Research and Development*, 31:464–474, 1987.

[40] G. J. Sussman. *A Computer Model of Skill Acquisition*. American Elsevier, New York, 1975.

[41] J.S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.

[42] M.M. Veloso and J.G. Carbonell. Towards scaling up machine learning: A case study with derivation analogy in prodigy. In Minton S., editor, *Machine Learning Methods for Planning and Scheduling*. Morgan Kaufmann, 1992.

[43] M. Waldrop. Will the Hubble space telescope compute? *Science*, 243:1437–1439, 1989.

[44] M. Zweben. A framework for iterative improvement search algorithms suited for constraint satisfaction problems. Technical Report RIA-90-05-03-1, NASA Ames Research Center, AI Research Branch, 1990.

[45] M. Zweben, M. Deale, and R. Gargan. Anytime rescheduling. In *Proceeedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*. Morgan Kaufmann Publishers, 1990.

A N-Queens conflict probability distributions

In this appendix we derive conflict distribution functions for the simplified n-queens model discussed in Section 5.3.2, which assumes that in the solution state exactly three other queens conflict with non-solution queen placements.

Consider first a non-solution value $Q_R^{\text{non-sol}}$ for a queen in row R. In the solution state there are three other queens which constrain $Q_R^{\text{non-sol}}$: denote this set by q. Let the number of queens other than R which have non-solution assignments be i. If R has a solution assignment, then $i = d$; and if R has a non-solution assignment, then $i = d - 1$. The probability of a conflict on $Q_R^{\text{non-sol}}$ due to a queen in q is:

$$p_1^q = \begin{bmatrix} \text{Prob. conflict on} \\ Q_R^{\text{non-sol}} \text{ from} \\ \text{queen in } q \end{bmatrix} = \begin{bmatrix} \text{Prob. queen in } q \\ \text{has non-solution} \\ \text{value} \end{bmatrix} \times \begin{bmatrix} \text{Prob.} \\ \text{non-solution} \\ \text{value conflicts} \\ \text{with } Q_R^{\text{non-sol}} \end{bmatrix} + \begin{bmatrix} \text{Prob. queen in } q \\ \text{has solution value} \end{bmatrix} \times \begin{bmatrix} \text{Prob. solution} \\ \text{value conflicts} \\ \text{with } Q_R^{\text{non-sol}} \end{bmatrix}$$

Now the probability that a queen in q has a non-solution value is $i/(n-1)$, and the probability that a non-solution value for a queen in q conflicts with $Q_R^{\text{non-sol}}$ is $2/(n-1)$ (i.e. two other placements would be either on the same row or diagonal as $Q_R^{\text{non-sol}}$). The probability that a solution value for a queen in q conflicts with $Q_R^{\text{non-sol}}$ is one by definition. Thus:

$$p_1^q = \frac{i}{n-1}\frac{2}{n-1} + \left(1 - \frac{i}{n-1}\right) = 1 - \frac{i(n-3)}{(n-1)^2}.$$

A similar argument leads to the probability of conflict with the $n-4$ queens *not* in q:

$$p_1^{\bar{q}} = \frac{3i}{(n-1)^2}.$$

The probability of v conflicts on $Q_R^{\text{non-sol}}$ is the sum of two binomially-distributed variables

$$P(v \text{ conflicts on } Q_R^{\text{non-sol}}) = \sum_{x=0}^{v} B(x, p_1^q, 3) B(v - x, p_1^{\bar{q}}, n-4),$$

assuming that the conflicts are independent. When there are no erroneous assignments, this distribution has a mean value of 3 and variance of zero, capturing the assumption that, in the solution state, each non-solution value has exactly three conflicts.

For a solution value Q_R^{sol} for a queen in row R, conflicts can arise only from non-solution assignments of the $n-1$ other queens. Assuming independence, the distribution of conflicts is:

$$P(v \text{ conflicts on } Q_R^{\text{sol}}) = B(v, p_2, n-1).$$

where $p_2 = 3i/(n-1)^2$.