Applications of Statistical Mechanics to Combinatorial Search Problems

Tad Hogg

Annual Reviews of Computational Physics vol. 2, pp. 357–406, World Scientific 1995

Abstract

The statistical mechanics of combinatorial search problems is described using the example of the well-known NP-complete graph coloring problem. We focus on a recently identified phase transition from under- to overconstrained problems, near which are concentrated many hard to solve search problems. Thus, a readily computed measure of problem structure predicts the difficulty of solving the problem, on average. However, this prediction is associated with a large variance and depends on the somewhat arbitrary choice of the problem ensemble. Thus these results are of limited direct use for individual instances. To help address this limitation, additional parameters, describing problem structure as well as heuristic effectiveness, are introduced. This also highlights the distinction between the statistical mechanics of combinatorial search problems, with their exponentially large search spaces, and physical systems, whose interactions are often governed by a simple euclidean metric.

Chapter 1

Introduction

Combinatorial search is among the hardest of common computational problems: the solution time can grow exponentially with the size of the problem [14]. Examples arise in scheduling, circuit layout design and automated proofs, to name a few areas. For instance, in scheduling classes, we attempt to find an assignment of rooms, times, students and faculty to each class to satisfy restrictions such as no two classes are in the same room at the same time. Similar scheduling problems arise in assigning tasks to machines in a manufacturing environment or experiments on spacecraft with constraints on power use. The circuit layout example involves placing devices on a circuit to minimize wiring or other costs. In a mathematical context, automated theorem proving can be viewed as an attempt to find a valid path from the axioms to the desired theorem. Combinatorial search also arises in discrete physical models such as finding the ground state of a spin glass system. A simple example of combinatorial search is the problem of coloring a square so that each corner has a different color from that of both nearest neighbor corners. This is readily done with two or more colors. However, a suitable coloring for larger and more complicated graphs is not so easily found because color choices that lead eventually to conflicts cannot always be recognized right away.

Fundamentally, these problems consist of finding those combinations or subsets of a discrete set of items that satisfy specified requirements. These satisfactory subsets are the solutions to the search problem. In the class scheduling problem, for instance, the discrete set could consist of all possible ways to assign classes to rooms, times, etc. and the task then viewed as selecting a subset of these possibilities that satisfies the requirements for the schedule. These problems are thus conceptually very simple to solve. Specifically, there are a finite number of possibilities so they can be examined until a suitable combination is found. Thus there are none of the issues of numerical precision, stability and roundoff errors encountered with, say, the numerical solution of differential equations. However, while conceptually simple, combinatorial search problems can be extremely time consuming to solve in practice. This is because the number of possible combinations to consider grows very rapidly (e.g., exponentially or factorially) with the number of items, leading to potentially lengthy solution times and severely limiting the feasible size of such problems. For example, a set consisting of n items has 2^n subsets, many of which may have to be examined to find a solution. This growth in computational cost is much faster than that for typical numerical algorithms such as the fast Fourier transform (cost of order $n \ln n$) or matrix multiplication (cost for the direct method of order n^3) where n characterizes the size of the problem: number of data samples for the Fourier transform, and number of rows or columns of the matrix for the linear algebra case. We should also note that combinatorial search problems can also arise in an optimization form: instead of searching for combinations satisfying specified requirements, one desires instead those combinations that minimize a given cost function. These forms are closely related both conceptually and in the methods used to solve them [14].

Even though the number of possibilities to consider grows rapidly, the actual number that must be searched before finding a solution can vary greatly depending on the order in which the possibilities are examined. In practice, heuristics [43] are often used to select the next combination, or state, to consider during a search among the possibilities. Typically, during a search the heuristic evaluates a small number of potential changes to the current state, based on easily computed local properties of the state, e.g., by considering only some of the requirements. The state with the highest evaluation is considered next. Often this leads to a series of choices that produces a solution much faster than uninformed random selection.

Sometimes, however, any such evaluation, based on local information, can be misleading with respect to the overall, or global, requirements.

When faced with such a search problem, it would be useful to know which heuristic is likely to be best, or determine whether the problem is solvable with available methods and current hardware speeds. This goal would be difficult to achieve if each problem or search method were very different from others. However, much recent progress has been made toward this goal through the use of a statistical mechanics approach. In this paper, we summarize these results for one type of combinatorial search, constraint satisfaction [35], using the particular problem of graph coloring as an example. Specifically, the next section describes the general reasons to expect useful insights from an approach based on statistical mechanics. Section three presents the constraint satisfaction problem. The following two sections summarize the phase transition identifying hard problem instances among randomly generated problems. Section six examines the robustness of this behavior for other problem ensembles. We then consider ways to use additional problem structure to more precisely characterize the search cost, an example of a deceptively hard problem and conclude with a discussion of a variety of open issues.

Chapter 2

Applications of Statistical Mechanics to Search

In a fundamental sense, the effectiveness of a search heuristic is determined by how the many repeated *local* decisions at each search step combine to determine the *global* behavior, i.e., the search cost. In detail this will depend on the specific problem and heuristic used. However, it is also useful to understand the typical behavior of general search methods for various classes of problems. This contrasts with the usual emphasis in computer science theory on a worst case analysis [14]. Focusing on typical behavior is particularly appropriate for evaluating search methods in practice since one usually is not interested in how well they work for a single given problem but rather for a variety of problems likely to be encountered in the future. In such cases, there may be only limited information available about the nature of the problems to be solved in which case the remaining details act as unspecified degrees of freedom and can be viewed as random variations.

In large physical systems with many unspecified degrees of freedom, statistical mechanics has been very successful at relating local and global behaviors of the system. So a natural question is how well it might describe large combinatorial search problems also having many unspecified degrees of freedom. The basic ingredients of this approach are to identify an ensemble of interest, i.e., a set of configurations and the probability each occurs, and then evaluate typical global behaviors of members of the ensemble. These typical behaviors will apply to most members when the variance is small. In statistical physics, the ensemble usually consists of all microstates consistent with given thermodynamic parameters and uses a simple probability distribution, e.g., all states equally likely in the microcanonical ensemble. Note that the choice of ensemble is somewhat arbitrary, e.g., the microcanonical or canonical ensembles in physics. This choice is based on several factors such as ease of use, the expectation that interesting real cases are typical within the ensemble rather than exceptionally rare, and a small variance in global properties at least as the size increases (i.e., in the thermodynamic limit).

In applying this approach to search problems, an ensemble consists of a class of search problems along with a probability for each problem in the class. Ideally one would like to select ensembles for search problems that have the broad range of applicability as found for those used in statistical physics. Unfortunately, the nature of realistic search problems is not yet well enough understood to make this possible. Instead, most work in this area uses simple random classes of problems, leaving to the future the identification of more realistic ensembles.

The use of statistical mechanics has been applied in a variety of situations to understand the generic behavior of combinatorial search problems [19]. These include phase transitions due to pruning in heuristic search [26], models of associative memory [29, 51], automatic planning [5], optimization problems [6, 64] and various cases of pattern matching [58, 23, 17, 33]. Here we will focus on the recent studies that relate search difficulty to the structure of constraint satisfaction problems [6, 9, 15, 32, 39, 47, 59, 60], with information on these results also available via the World Wide Web [21]. Specifically, for large problems, a few parameters characterizing the structure of the search problem determine the difficulty for a wide variety of common heuristics, on average. Moreover, changes in these parameters give rise to transitions, becoming more abrupt for larger problems, that are analogous to phase transitions in physical systems. These identify situations in which major gains are possible from small improvements in the local heuristic evaluations. This work is also beginning to lead to improved search methods [8, 65] as well as a refined understanding of the structure of the search problems [22].

Chapter 3

Constraint Satisfaction Problems

In this paper we focus on a particular kind of combinatorial search: the simple *constraint satisfaction problem* (CSP). This consists of n variables each of which can be assigned one of b values, and a set of constraints that restrict allowable combinations of assignments. There are a variety of ways to characterize a constraint. Most fundamental are the smallest individual partial assignments that violate the constraint. In this context, a *partial assignment* is a set of assignments to some of the variables in the problem, not necessarily all of them. Its size is just the number of variables assigned a value. These smallest conflicting assignments for all the constraints are constitute the set of *minimized nogoods* for the problem. Often in describing a set of constraints, it is sufficient to specify the size and number of such nogoods rather than their detailed structure [62].

In this context, the search problem is then to find a solution, i.e., an assignment of a value to each variable such that all the constraints are satisfied, or else show that no such assignment exists. In set-theoretic terms, a solution is a complete assignment (i.e., a value for each variable) that is not a superset of any of the minimized nogoods specified by the constraints. To connect with the general combinatorial search problem, finding a consistent assignment can be viewed as selecting from the set of all possible variable-value pairs a subset such that each variable appears exactly once and all of the constraints are satisfied. There are nb such variable-value pairs and hence a total of 2^{nb} possible subsets. However, the requirement that each variable is given just one assignment means we can restrict the search to the total number of ways to assign values to the variables, i.e., b^n, which grows exponentially with the size of the problem, n.

In this section we describe a variety of search methods used to solve constraint satisfaction problems and present the specific example of graph coloring used to illustrate the behaviors discussed in the remainder of the paper.

3.1 Search Methods

While there are a large variety of search methods for CSPs [48], they can be conveniently viewed as one of the following strategies (which also readily generalize to other combinatorial search problems). First and simplest is to *examine* all possibilities in turn until a solution is found or no further possibilities remain. While conceptually straightforward, this method of *generate and test*, is quite slow unless most possibilities are in fact solutions or a clever ordering heuristic makes it likely that a solution will be found early in the enumeration of the set of possibilities. At worst, when there are no solutions, this method will examine all b^n possible assignments.

The second strategy attempts to *construct* a solution from partial solutions, i.e., assignments to some of the variables that are consistent with all the constraints. Generally this operates by attempting to add a consistent assignment to a currently unassigned variable, and backtracking when that is not possible. The general backtrack procedure backtrack(*state*, *d*) operates on a state consisting of a consistent partial assignment to d of the variables and can be described as follows:

```
if ($d$ equals $n$) then
  return state as a solution
```

```
else
  pick an unassigned variable $v$ in $state$
  for each value $i$ that is consistent for $v$
    $s$ = $state$ with $v$ assigned value $i$
    $result$ = backtrack($s$, $d$+1)
    if ($result$ is a solution) then
      return $result$ as a solution
    endif
  endfor
  return ‘‘no solution’’
endif
```

The search begins with a completely unassigned state and with d, the number of assigned variables in the state, set to zero. The worst situation for this search method is when there are no solutions but every partial assignment with a single unassigned variable is consistent with the constraints. In this case the search will consider all of the b^d states with d assignments (in a given variable ordering) for $0 \leq d < n$ a total of $(b^n - 1)/(b - 1) \sim b^n/(b - 1)$. More typically, when each constraint involves only a few variables, many of the partial assignments will have no consistent extensions so that large parts of the search space need not be considered at all. So we can expect backtracking to be effective either when there are many solutions so backtracking happens only rarely, or the constraints rapidly eliminate unproductive search paths (i.e., those partial assignments that are not subsets of a solution) because they violate a constraint as soon as a few of the wrong choices are made.

Note the recursive nature of this backtrack procedure, which operates in a “depth-first” manner in that it attempts to assign as many variables as possible before backtracking. This has the advantage of requiring a minimal amount of memory to store the current search state. Another variation, “breadth-first” search, considers all states with a single assignment before moving on to those with two assignments, etc. This can be appropriate for search problems, such as automatic theorem proving, for which the number of steps to a solution is not known initially, but it also requires additional memory to store the pending search states at each level of the search.

Backtrack search to construct full solutions from partial ones has two natural points for the use of heuristics. First is the selection of the next unassigned variable to consider. Ideally one would select next the variable that is most likely to cause a conflict so that if this search path is ultimately unproductive this can be determined as soon as possible. (On the other hand, if this search path does lead to a solution, all variables will need to be considered eventually anyway so nothing is lost in always trying to select the variable most likely to give a conflict.) A simple, and often effective, way to do this is to pick next the variable with the fewest remaining available assignments. The second place for using a heuristic is in the order of assigning the values to the new variable. Here we would like to first try values most likely to lead to a solution, if any, consistent with the current partial assignment. If successful, this eliminates the need to consider additional values for the current variable. Here a useful method is to select the least constraining values first in the hope that they will leave open many options for the remaining unassigned variables. Conceptually, these heuristics could determine the best possible choice by complete search. But this would of course require as much computational effort as doing the search directly. So instead the heuristics are further restricted to be able to execute rapidly. Thus, in practice, these heuristics evaluate potential choices by the use of local information only, i.e., how the choices directly affect other variables, but do not consider how those effects may subsequently affect future choices and so will not always give the best possible choice.

In addition, when forced to backtrack, one can note the previous variable assignments that were responsible for causing the conflict. Then, instead of simply backtracking to the most recent assignment (so-called *chronological* backtrack) one can instead backtrack to the most recently assigned variable actually involved in the conflict.

The third general search strategy is to work with a complete state (i.e., a state in which each variable is assigned a value) and attempt to *repair* it by a series of changes to the assignments to remove conflicts, eventually giving a solution. For computational simplicity, each of these changes is local, i.e., a relatively small modification to the state. For the common case where changes consist of reassigning a single variable at a time, the general procedure repair($state$) can be described as follows:

```
while (state is not consistent)
  pick a variable $v$ to change
  pick a new assignment for variable $v$ in state
endwhile
return state as a solution
```

The search begins with an initial complete state, often selected at random. Since it may be the case that a solution cannot be reached from the initial state through a series of local changes, in practice, this method is usually restarted from a new initial condition if it has not found a solution after a prespecified number of steps. Usually this new initial condition is chosen randomly, but it can also be based to some extent on the remaining conflicts from the previous search attempt [49]. Note that, unlike the previous search strategies, this method will never terminate if there is no solution for a problem, i.e., it is not a systematic or complete search method. Thus another bound is introduced in practice to limit the number of trials, i.e., restarts with new initial conditions. After reaching this bound the search terminates unsuccessfully. Nevertheless, the repair strategy can often be much more effective than complete methods such as backtracking when a problem is in fact soluble. It is also suitable for dynamic problems in which the constraints change over time (e.g., in a class scheduling problem in which a class is canceled or a new section added). In these cases one can often start from an available solution to the initial CSP and repair it to rapidly find a new solution as the CSP changes.

With this repair method, heuristics can be used to guide the selection of the next variable to consider and the new value for its assignment. A simple possibility is *hillclimbing* in which a change that reduces the number of remaining conflicts is selected randomly. This is analogous to searching for an energy minimum in a discrete physical system. Alternatively, one can choose any variable whose assignment conflicts with a constraint and set its new value to the one that minimizes the number of conflicts [38], or select the variable to change that reduces the remaining conflicts as much as possible [50]. This repair method can get stuck in "plateau" states, in which no additional single assignment change reduces the number of conflicts. A common way to address this possibility is to allow for changes that leave the number of conflicts unchanged. This then amounts to a walk among plateau states with the same number of conflicts until a new state is found that allows the search to proceed to fewer conflicts. Because the heuristic of minimizing conflicts is unable to provide guidance for selecting new states to try in this situation, moving on this plateau essentially amounts to a random walk which can take a long time. Even worse is the possibility of local minima, states in which any single change increases the number of conflicts. In these cases, even allowing moves to other states with the same number of conflicts will not help. This then requires either restarting the search or allowing occasional increases in the number of conflicts as is done with the simulated annealing search method [30]. From this discussion, we can see that the repair strategy is likely to be useful when there are few local minima and small plateaus so the heuristic guidance of locally reducing the number of remaining conflicts is likely to lead to a solution (i.e., a global minimum). Otherwise the search will require either many restarts or trying to escape from the plateaus and local minima using some sort of random repair choices which will not have the benefit of the heuristic guidance. In either case the search is likely to require many steps and may never find a solution within the prespecified bounds. The density of the plateau states and local minima within the search space is thus an important characteristic for the success of this method, and has been investigated in the context of some simple constraint problems [40, 62].

Further improvements in all these search methods are often possible by identifying and removing trivial aspects of the problem before searching, checking for whether the problem can be decomposed into independent parts, and using the solution of a simplified, coarse-grained or abstract version of the problem as a guide to finding the detailed solution. There is also the possibility to avoid unnecessary repetition of parts of the search by recording those combinations found not to work [10]. However, with all these heuristics it is important to realize the extra use of computational resources (both time and memory) may counteract the savings in search steps. Furthermore, in some cases they can result in an increase in the required search steps [45, 1] when the nature of the problem strongly violates the assumptions of the heuristics.

3.2 Graph Coloring

As a specific example of a constraint satisfaction problem, in this paper we use the well-studied graph coloring problem. In this context, a *graph* consists of a set of *nodes* and a set of *edges* that link some pairs of these nodes. We consider only simple graphs in which edges link distinct nodes and a pair of nodes has at most one edge between them. A *graph coloring problem* consists of a graph, a specified number of colors, and the requirement to find a color for each node in the graph such that adjacent nodes (i.e., nodes linked by an edge) have distinct colors. Note that this problem has a special symmetry: if a given assignment of colors is a solution, additional solutions can be obtained simply by permuting the colors. As a well-known NP-complete problem [14], graph coloring has received considerable attention [38, 27, 50]. That is, graph coloring is among the class of hardest combinatorial search problems and so provides a particularly challenging example. Specifically, this means that a general, rapid search method for graph coloring could be modified to also solve any other combinatorial search problem with relatively little additional computational effort. Thus the behaviors observed for graph coloring can be expected to occur with other search problems as well. In this context, a rapid search method refers to one whose computational cost grows as a polynomial in the size of the problem, n, despite the fact that the number of possible assignments grows exponentially fast. Whether such rapid methods exist in general is an important open question in computer science theory.

Many important artificial intelligence problems, such as planning and scheduling [63, 2], can be mapped onto the graph coloring problem. To see how this can be done, consider the problem of scheduling n classes in b time slots. We can represent the classes as nodes in a graph. Each edge then represents the constraint that the two classes it links should not be scheduled at the same time (perhaps because they are taught by the same person or require use of the same lab equipment). Finding an acceptable schedule then amounts to assigning a time to each class such that classes connected by an edge in the graph are given different times. This is just the graph coloring problem.

In terms of the general constraint satisfaction problem described above, the n nodes in the graph are the variables, the b available colors are the possible values for the variables and each of the e edges of the graph is a constraint, requiring that the two variables linked by that edge have different assigned values. For simplicity, we primarily consider the ensemble of problems given by random graphs [3], i.e., taking each graph with a specified number of nodes and edges to be equally likely.

Our experiments primarily used a depth-first backtracking search based on the Brelaz heuristic [27]. This heuristic determines the order in which variables are assigned as well as the order in which values are tried. Specifically it assigns the most constrained nodes first (i.e., those with the most distinctly colored neighbors), breaking ties by choosing nodes with the most uncolored neighbors, and with any remaining ties broken randomly. For making color assignments to the selected node, this heuristic uses an arbitrary but fixed ordering for the colors $1, 2, \ldots, b$. For each node, colors are considered according to this ordering: the earliest color in this ordering consistent with the previous assignments is chosen first, with successive choices made when the search is forced to backtrack. The motivation here is to try to use as few colors as possible when a choice is available, hopefully leaving the remaining colors for future, highly constrained nodes. As a simple optimization, we never change the colorings for the first $b - 1$ nodes selected by this heuristic. Any such changes, which could only occur when the backtrack search has failed to find a solution starting from the initial assignments for these initial nodes, would amount to unnecessarily repeating the search with a permutation of the colors.

We measure the search cost by the number of states that are expanded until the first solution is found or, when there are no solutions, until no further possibilities remain to be examined. The search routines were written in C and were not fully optimized. The execution time required for each search step was roughly proportional to the number of edges. For example, on 100–node graphs our program performed about 26000 and 18400 steps/second on graphs with 150 and 225 edges respectively. As described below, some of the searches first decomposed each graph into independent components which were then searched separately. For 100–node graphs, the decomposition required about 0.2 seconds per graph, and the search steps ran a bit slower due to using component information during the search. All these execution times were on a Sun SPARC 10.

We have also experimented with a heuristic repair strategy. Specifically, at each step a variable whose assignment conflicts with a constraint was selected randomly and its value was set to the one that minimizes the number of conflicts [38]. Note that even if the state can be improved by a single assignment change, the random selection of the variable to change could select a variable for which no change improves the state. This allows new assignments that result in no change in the number of conflicts, but never for an increase. Thus this search method can proceed, although without heuristic guidance, among states with the same number of conflicts, but will be stuck in any local minimum encountered. To allow for this possibility, the search from each random initial condition was terminated if no reduction in the number of conflicts was found after a prespecified number of steps.

3.3 Search Examples

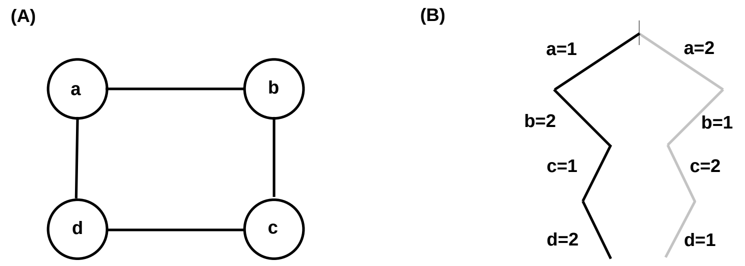


Figure 3.1: A) A graph with 4 nodes, labeled a, b, c and d. B) Example search tree to color the graph using two colors. Each level in the tree corresponds to a consistent color assignment for a particular node in the graph, indicated next to each link. Thus for example, $a = 1$ means color number 1 is assigned to node a. The black links indicate the part of the search tree that is examined to find the first solution. The gray links indicate the remaining part of the tree required to find all solutions. In this search tree, assigning color 1 or 2 to a node is represented by a link that goes to the left or right, respectively. This choice for the horizontal layout of the search tree means that, when colors are tried in numerical order, the tree is examined top to bottom and then left to right. The specific horizontal location also centers each choice in the search tree above the location of all choices for the remaining nodes in the graph (although only those choices that are consistent colorings are shown in the figure).

To make these algorithms more concrete consider the simple graph in Fig. 3.1A. This graph has four nodes and four edges forming a cycle. The behavior of a backtrack search can be described by the search tree representing the partial colorings it considers. Starting at the top with no assigned colors, each level in the tree corresponds to an attempt to color an additional node in the graph. In this example the nodes are colored in the order a, b, c and d. In this case, once the first node is assigned, there is only one remaining color for each successive node considered. This leads to the solution $(a = 1, b = 2, c = 1, d = 2)$ without any backtracking, as illustrated by the solid links in the search tree. Thus the solution is found in 4 steps. There is also a second solution for this problem shown by the gray links in the tree. Together, the black and gray links show all the consistent partial solutions obtained with this ordering of the nodes.

This simple case also provides an example for heuristic repair. Suppose we start with the state $(a = 2, b = 2, c = 2, d = 2)$ in which all nodes are assigned the second color. In this state, every node is in conflict with two other nodes. Suppose we select node a to change. The assignment $a = 1$ removes the two conflicts for this node, and is the minimum conflict assignment. This gives $(a = 1, b = 2, c = 2, d = 2)$ in which node a is not in conflict, nodes b and d have one conflict each and node c has two conflicts. Thus the next step of the repair procedure will randomly select one of the conflicting nodes for the next change. Suppose node d is selected next. In this case changing the assignment to d removes one conflict but introduces another (with node a). So both color choices result in no net change in the number of conflicts. In our implementation, the selection among these equivalent options was made randomly. For the sake of this example, suppose no change was made. For the third step we still have the same three conflicting nodes to choose among. Suppose node c is selected. With $c = 1$ we remove all conflicts, finally producing a solution.

For comparison, Fig. 3.2 shows an example with no solution. Again the nodes are considered in the order a, b, c and d. In this case however, once consistent colors are found for the first three nodes, there is no consistent way to color the fourth node. This forces the search to backtrack, eventually exhausting the possibilities and concluding there are no solutions for this search.

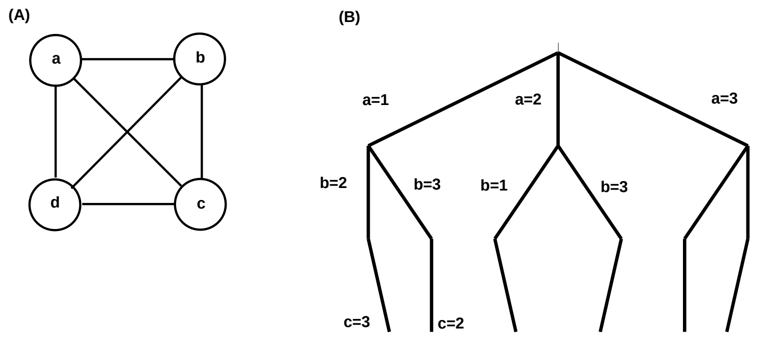


Figure 3.2: A) The complete graph with 4 nodes, labeled a, b, c and d. B) Example search tree to color the graph using three colors. Each level in the tree corresponds to a consistent color assignment for a particular node in the graph, indicated next to some links. Thus for example, $a = 1$ means color number 1 is assigned to node a. The black links indicate the part of the search tree that is examined to determine there are no solutions.

Chapter 4

Phase Transitions in Search

As mentioned above, there are by now a variety of situations in which phase transition-like behaviors have been observed in combinatorial search problems. Here we review a particular case in which simple measures of problem structure determine, on average, the search cost with a variety of search methods.

4.1 Problem Structure and Search Cost

For graph coloring, the average degree of the graph γ (i.e., the average number of edges coming from a node in the graph) is a parameter that distinguishes relatively easy from harder problems, on average. This parameter, also called the connectivity, is related to the number of edges e and number of nodes n in the graph by $e = \frac{1}{2}\gamma n$. In this paper, we focus on the case of 3–coloring (i.e., when 3 different colors are available).

4.1.1 an example

To gain an understanding of how the number of edges determines the search cost, we first consider a small example. Specifically, consider a series of connected graphs constructed from a random tree [41] with 10 nodes (and 9 edges). Additional edges were added randomly to give a series of related graphs. Each graph was then searched using a simple, nonheuristic backtrack search in which nodes were colored in numerical order (for graphs this small, the Brelaz heuristic is usually able to find a solution without any backtrack, if one exists, and so does not provide an interesting illustration of the behavior).

The resulting behavior is shown in Fig. 4.1. This shows that the sparse and dense graphs are relatively easy to search, while intermediate ones give rise to harder searches. This behavior can be readily understood from the changes in the search trees as edges are added, as shown for a few cases in Fig. 4.2. When there are few edges, there are a large number of solutions and one of them is quickly found, without any need for backtracking. As more edges are added, solutions are rapidly eliminated, while partial states with fewer assignments are removed more slowly. This results in a search tree in which many states at intermediate levels do not lead to solutions, increasing the need for backtracking. At some point, the last solutions are eliminated, giving a large increase in the search cost as the backtracking must now continue to check all possibilities. At this point we say the problem is critically constrained in that there are just enough constraints to eliminate the solutions. Beyond this point, additional edges continue to prune the intermediate states in the tree which results in a decreased search cost. This discussion also suggests why the maximum search cost is seen just as the last solutions disappear.

4.1.2 example: soluble graphs

This easy-hard-easy behavior is also seen when the graphs are constrained to have a solution, as shown for another set of random graphs in Fig. 4.3. These soluble graphs were generated by randomly creating graphs with 29 edges until one with a solution was found. Edges were then randomly deleted to produce the remaining graphs. Note that deleting edges preserves the property of having a solution.

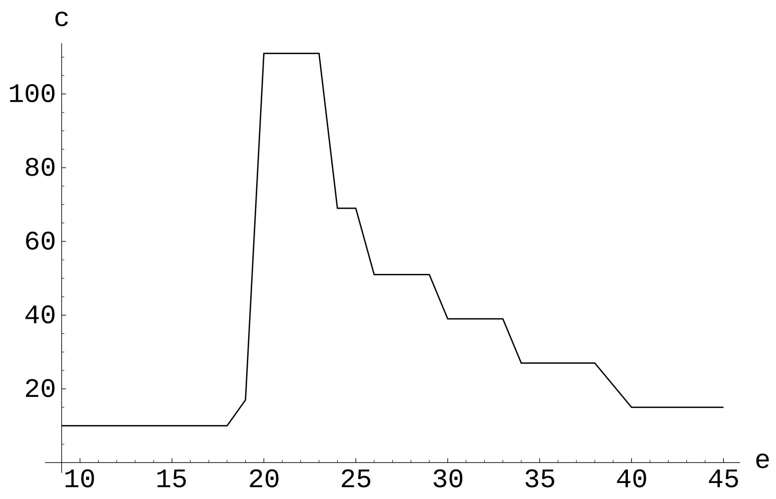


Figure 4.1: Search cost c to find the first solution, if any, or determine there are none for 3–coloring a randomly generated set of connected graphs with 10 nodes, vs. the number of edges e in the graph. The search used nonheuristic backtrack. The graphs range from a random tree, with 9 edges, to a complete graph (each node linked to every other node) with 45 edges. A solution exists for those cases with 19 or fewer edges. Thus the peak in search cost occurs at the point where the solutions just disappear.

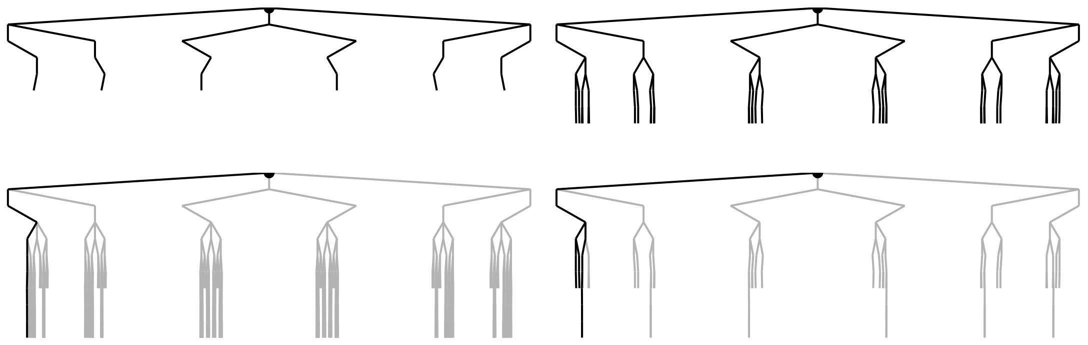


Figure 4.2: Search trees. Starting from the upper left and continuing clockwise these correspond to the graphs with 35, 20, 19 and 12 edges respectively. Black lines show the states searched to find the first solution or determine there are none. Gray lines show additional consistent partial states.

Examining the search trees for this class of graphs again shows the competition between the rapid pruning of solutions and the slower pruning of intermediate states. As before, this initially gives rise to a general increase in search cost as edges are added. Eventually only one solution remains (up to a permutation of the colors) so no further pruning of the solutions can take place (because these graphs are constrained to have at least one solution). However additional edges continue to prune the intermediate states, giving a reduction in the search cost. This shows the notion of critically constrained problems applies more generally than just cases in which the last solution disappears.

For this set of soluble graphs we can also consider the behavior of an incomplete search method such as heuristic repair, as shown in Fig. 4.5. Since this search method is inherently random, we repeat the search to give an estimate of the typical behavior. Qualitatively, we see the same behavior as with the backtrack search method: with few or many edges the search is easy, while intermediate cases are hard.

This behavior can be partly understood by examining the local environment of each of the 3^{10} possible complete assignments for these graphs. Specifically, we can expect the search to get harder, on average, when there are fewer solutions. Similarly, a high density of difficult states should also result in longer search times. For heuristic repair, difficult states are those for which local changes do not decrease the number of conflicts.

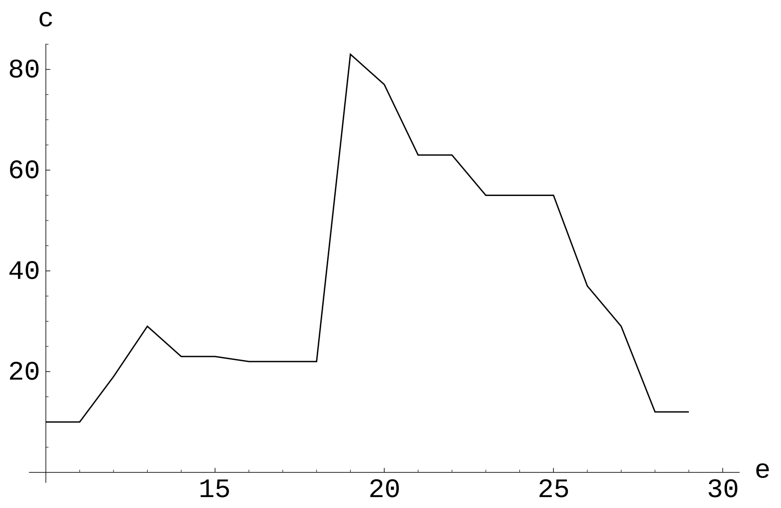


Figure 4.3: Search cost c to find the first solution for 3–coloring a randomly generated set of soluble graphs with 10 nodes, vs. the number of edges e in the graph. The search used nonheuristic backtrack. Multiple distinct solutions exist for those cases with 18 or fewer edges. Thus the peak in search cost occurs at the point where the last extra solution just disappears.

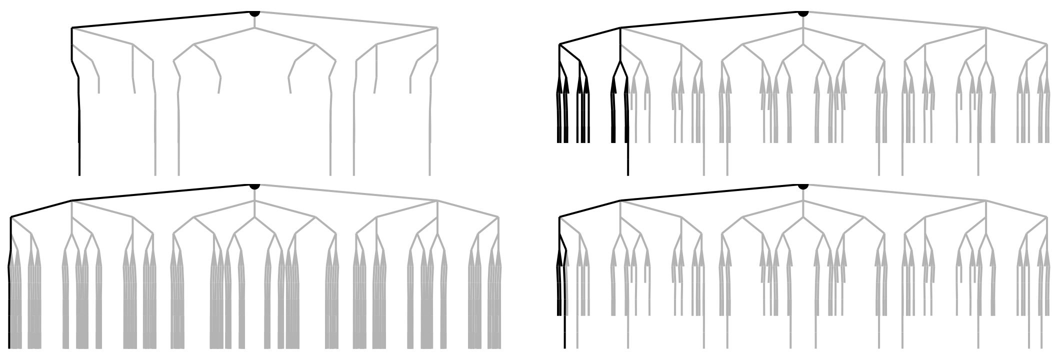


Figure 4.4: Search trees. Starting from the upper left and continuing clockwise these correspond to the graphs with 29, 19, 18 and 11 edges respectively. Black lines show the states searched to find the first solution. Gray lines show additional consistent partial states.

One way to characterize this behavior is to count the number of nonsolution plateau and minimal states. Here a minimal state is one for which all local changes, i.e., changing the assignment to a single node, result in more conflicts. This heuristic repair method will remain stuck in a minimal state if it ever reaches one, until the limit on successive unimproved steps is reached and the search restarts from a new initial condition. Plateau states, where no single change can decrease the number of conflicts but there are changes that give the same number, are not necessarily fatal to the search: it may be possible to move to another state with the same number of conflicts which then allows for a decrease, i.e., a chance to escape from the plateau. Even if this is possible, it can require a long random walk to find a way off a plateau and, even worse, there may be no way to escape. The behavior of these quantities is shown in Fig. 4.6. As edges are added we see a generally exponential decrease in the number of solutions, along with a rise in the number of plateau and minimal states, which reach a peak at about the same point as the last extra solution disappears. The combination of these factors suggests that searches of such graphs will be particularly difficult, as confirmed in Fig. 4.5. As additional edges are added, the number of solutions remains constant but the number of difficult states drops, giving rise to easier searches.

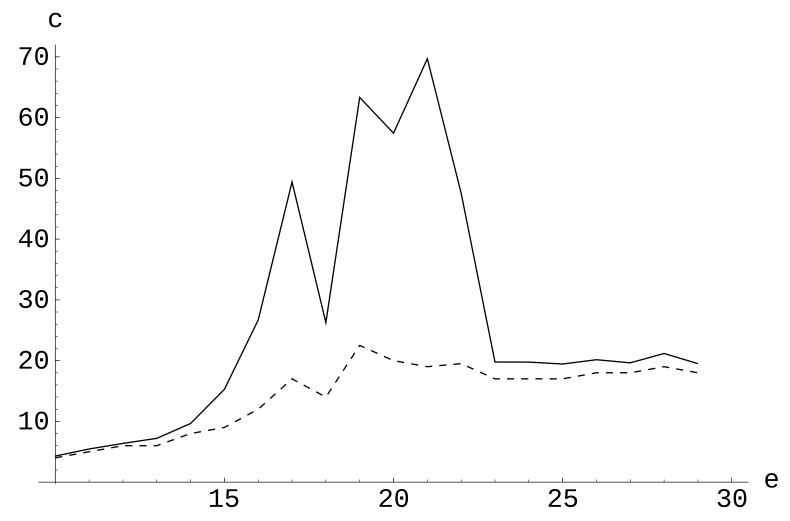
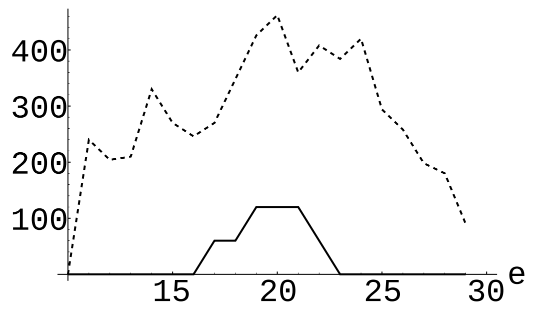


Figure 4.5: Mean (solid) and median (dashed) cost to find a solution with heuristic repair, starting from a random initial state. Each graph was searched 500 times using a limit of 100 unimproved steps, and up to 1000 trials. With these limits, all the searches were able to find a solution.



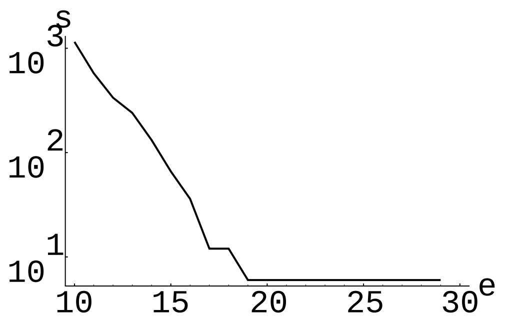


Figure 4.6: Number of search states of different types vs. number of edges in the graph. On the left is the number of nonsolution minimal states (multiplied by 10 for clarity) as the solid curve, and the number of nonsolution plateau states (dashed). Note that minimal states exist only for graphs with between 17 and 22 edges, inclusive, and in fact have only a single conflict. On the right is the number of solutions on a log scale.

4.1.3 average behavior

While the small search examples presented above give insight into the nature of the transition, its significance only becomes apparent when we examine many larger examples. The relation between the graph's structure and the number of search steps required to color it, or determine no coloring is possible, is shown in Fig. 4.7. Specifically, this shows that sparse and dense graphs are typically easy to color while those with an intermediate number of edges are more difficult. The observed peak for random graphs is at $\gamma = 4.6$. As shown in the figure, this peak also coincides with the point at which the fraction of graphs with a solution drops from near one to near zero. For graphs with more nodes, the peak in search cost becomes sharper and the fraction of soluble cases drops more abruptly. While this behavior with increasing size has primarily been studied empirically, finite-size scaling [31] and approximate "mean-field" theories [62] have also been applied. However, a definitive theory of this transition, similar to the rigorous thresholds known to exist for other properties of random graphs [3], still remains an open issue.

This observation associates a region with a high density of relatively hard problems with an abrupt transition in the nature of the problems themselves. That is, the drop in the fraction of soluble cases represents a transition from underconstrained graphs with many solutions to overconstrained ones with

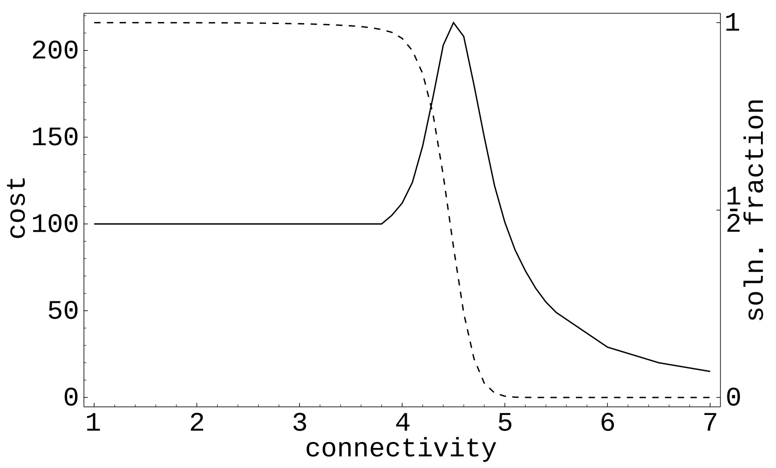


Figure 4.7: Behavior for 3–coloring of random graphs with 100 nodes as a function of connectivity γ in steps of 0.1 using backtrack search with the Brelaz heuristic. The solid curve shows the median search cost, and the dashed one is the fraction of graphs with a solution (ranging from one on the left to zero on the right), based on 50,000 samples at each point.

none. This relationship between search cost and problem structure is important since it suggests there is a fundamental aspect of the problems themselves that gives rise to difficult searches, rather than being particular to specific search methods. Similar behavior is also seen for heuristic repair as shown in Fig. 4.8.

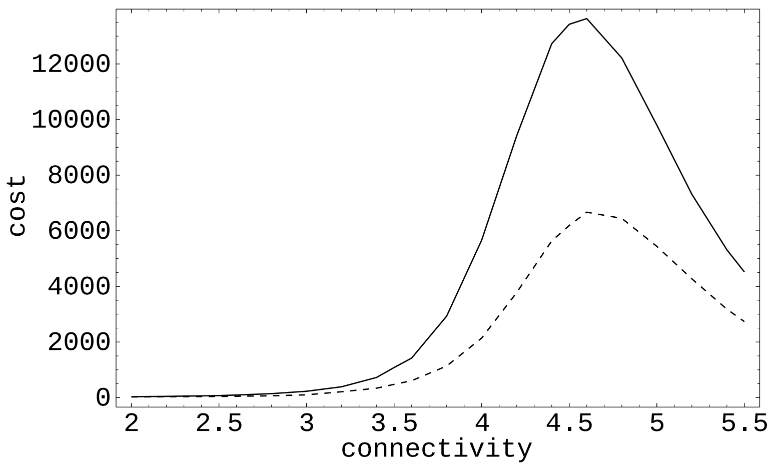


Figure 4.8: Mean (solid) and median (dashed) cost to find a solution with heuristic repair for 50–node soluble graphs as a function of connectivity. Each search was restarted from a new random initial condition after 100 successive steps failed to reduce the number of conflicts. This was continued until all searches completed.

In the transition region, graphs typically have many large partial solutions (i.e., consistent ways to color large subsets of the graph) but few, if any, complete solutions. This means that backtrack search has many opportunities to select unproductive paths, and that many of those paths will not be recognized as such until many incorrect assignments have been made (these lengthy unproductive paths correspond to the many large partial solutions that cannot be extended to full solutions). Similarly for repair-based searches, we can expect the large partial solutions to give rise to many local minima. This is because most of the many large partial solution will have little in common with the rare complete solutions. In such cases, any change that removes a conflict among the few remaining variables is likely to introduce one or more conflicts among the many variables that are not in conflict. These arguments suggest that the observations of difficult searches can be understood by examining the nature of the partial solutions, without requiring a detailed analysis of the individual search methods.

4.2 Theory

This behavior can be quantified with the following approximate argument [60, 54]. Let b be the number of available colors. A given edge in the graph eliminates b of the b^2 possible ways to color the nodes it connects. So a choice of colors will satisfy the constraint with probability $1 - 1/b$. Consider a state in which k nodes of the graph are colored. There are b^k possible colorings for this subgraph, and the probability a given edge will be within this subgraph is $\binom{k}{2}/\binom{n}{2} \sim \left(\frac{k}{n}\right)^2$. Thus a coloring of the k nodes will violate the constraint of a given edge with probability $\left(\frac{k}{n}\right)^2 \frac{1}{b}$. Assuming independence among the coloring constraints of the different edges, there will be, on average,

$$N_k = b^k \left(1 - \left(\frac{k}{n}\right)^2 \frac{1}{b}\right)^e \tag{4.1}$$

consistent colorings for these nodes.

The behavior of N_k, shown in Fig. 4.9, qualitatively explains the search behavior of Fig. 4.7. Specifically, each step of a backtrack-based search method, as used here, attempts to extend a partially colored subgraph to consistently include one more node, backtracking when there are no available colors for the new node. From the figure, we see that when there are few edges, N_k increases monotonically, so on average there will be at least one consistent way to color the next node considered. This means that a typical backtrack search will almost always be able to extend the partially colored subgraph, and go directly to a solution with little or no backtracking. As more edges are added, the number of solutions, N_n, decreases very rapidly, while the number of partial colorings for smaller parts of the graph decreases less rapidly. This leads to a maximum in N_k at a value $k_{\max} < n$. In this situation, a typical search will proceed with little backtrack up to $k \approx k_{\max}$ but is then unlikely to be able to proceed further. Thus we can expect a great deal of backtracking until the search finds one of the relatively rare partial colorings that does lead to a solution. This leads to an increase in search cost, which continues as long as the number of solutions drops more rapidly than the number of partial solutions at $k_{\max}$. Finally, when there are very many edges, there are unlikely to be any solutions in which case all partial solutions must be examined in the search. However, as seen in Fig. 4.9, the number of partial solutions, $\sum_k N_k$, decreases as edges are added resulting in a lowered overall search cost.

This qualitative description of the behavior of the search cost predicts that the hardest problems will occur at about the same point as when the number of solutions finally drops to zero, explaining the correspondence between the search cost peak and the drop in the probability for a solution to exist. Another observation is that when there is little or no backtracking (i.e., when N_k is growing monotonically), the search cost should grow only linearly with the size of the problem, n. Otherwise, we can expect exponential growth in the search cost as n increases, due to the exponentially increasing difference between the number of partial colorings, especially $N_{k_{\max}}$, and the number of solutions.

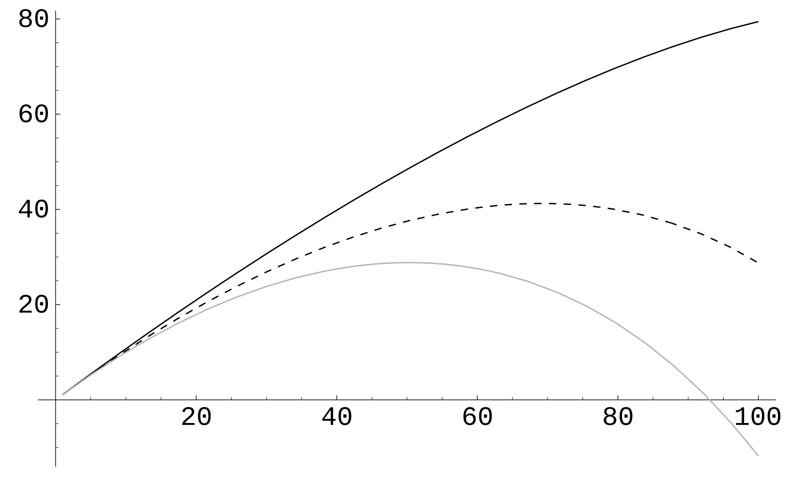


Figure 4.9: Number of partial colorings as a function of their size. Specifically, this shows $\ln N_k$ vs. k for $n = 100$ and $b = 3$ for $\gamma = 1.5$ (solid), 4 (dashed) and 6 (gray).

We can also use Eq. (4.1) to estimate the regions with these different behaviors. First, N_k increases monotonically as long as $\gamma < (b-1)\ln b$, or 2.2 for $b=3$. Second, a rough criterion for when the number of solutions just reaches zero is $N_n = 1$, i.e., at $\gamma = \frac{-2\ln b}{\ln(1-1/b)}$, or 5.4 for $b=3$. For larger or smaller γ, N_n approaches zero or infinity, respectively, as n increases.

In addition, for sufficiently large γ the peak in the number of partial states will be at $k_{\max} \ll n$ giving a regime of polynomial growth in the search cost. In this regime

$$\ln N_k \sim k \ln b - e\left(\frac{k}{n}\right)^2 \frac{1}{b} \tag{4.2}$$

with maximum at $k_{\max} = \frac{n}{\gamma} b \ln b$, and $\ln N_{k_{\max}} \sim \frac{n}{2\gamma} b (\ln b)^2$. Thus we can expect polynomial search cost when this grows logarithmically, i.e., when $\gamma = O\left(\frac{n}{\ln n}\right)$ (corresponding to edge probability of $\frac{\gamma}{n} = O\left(\frac{1}{\ln n}\right)$) This suggests that dense graphs are easy to color, or determine that no coloring exists, which is also the case when restricting attention to graphs that do have a coloring [57, 12].

To summarize, this theory shows how the variation in search cost as edges are added is due to a competition between increased pruning and a reduction in the number of solutions. Additional edges reduce the number of solutions much more rapidly than they increase pruning of partial colorings. This results in the following behavior. For very sparse graphs, there are so many possible solutions that one is easily found, in polynomial time on average. When the connectivity exceeds a critical value, the cost grows exponentially. The rate of exponential growth increases until all solutions disappear. Beyond this point, the increased pruning reduces the exponential growth rate of the cost, so the point where the last solutions disappear corresponds to the highest cost. Finally, when the connectivity exceeds a higher threshold, the search cost becomes polynomial again.

This theory makes two simplifications. First, it ignores edge dependencies, leading to relatively small errors in specifying the location of the transition, which can be improved further by incorporating some of these dependencies [60]. Second, the theory does not account for the ability of heuristics to color nodes in an order that increases the likelihood of early pruning, leading to substantial differences in the actual cost estimates between the theory and various heuristic search methods. Despite these limitations, the theory describes the essential mechanism driving this transition from under- to overconstrained problems and explains why hard cases are likely to be found in the transition region. In more quantitative terms, it predicts an observed transition from linear to exponentially growing search cost [25] and determines roughly the parameter values at which these transitions take place. In addition, the peak in the search cost also appears [60] for methods, such as heuristic repair [38] or simulated annealing [30], that incrementally modify a completely colored graph in an attempt to remove conflicts. For these methods the peak in search cost is due to changes in the relative density of solutions among the complete colorings whose number of conflicts cannot be reduced by a single change.

4.3 Applications

These observations are encouraging progress in characterizing combinatorial search. Specifically, relating simple, easily-computed structural parameters of the problem to the behavior of a variety of search methods gives the possibility of readily evaluating problem difficulty and hence may offer some guidance in selecting among different formulations of a search task. This is particularly true in light of the fact that similar behavior is seen with other search methods and for other constraint satisfaction problems. However, the situation is in fact more complicated and further refinements are needed. In particular, this discussion applies *on average* but there is substantial variation among the individual problems and different search methods. Moreover, the precise location of the transition point depends to some extent on the class of problems considered. These behaviors make it difficult to apply these results to confidently predict the behavior of individual search problems.

Nevertheless, there are ways to apply these observations. The first is simply to note the transition region is useful to generate search problems that are likely to be difficult for a variety of search methods. They can thus be used as a readily generated standard set of test cases to compare different algorithms.

More interestingly, this phenomenon suggests some possible improvements in the search methods themselves. In particular, it is precisely when individual search methods become ineffective that the overhead associated with more sophisticated methods may be worth incurring. This has led to studies of cooperative search for graph coloring [24]. More generally, because this phenomenon relates likely search difficulty to easily computed structural parameters, it can be used as a heuristic when selecting the next subproblem to work on [8].

Chapter 5

The Search Cost Distribution

The previous section described an easy-hard-easy transition in search cost and gave a simple theoretical explanation for this behavior. However, a more complex story is seen from the full distribution of search costs. Surprisingly, as shown in Fig. 5.1, exceptionally hard instances are concentrated not around the peak in the median, but rather at lower connectivities [25]. Thus, for 100–node graphs, $\gamma = 4.5$, near the median peak, gives many more cases with cost above 1000 than $\gamma = 3$, but the reverse is true for costs above 100,000. This suggests that there are actually two qualitatively distinct regions of hard problems: 1) a region containing a high density of relatively hard problems giving the peak in the median, and 2) a region, with somewhat lower connectivity, in which most problems are very easy but which also contains a few exceptionally hard instances.

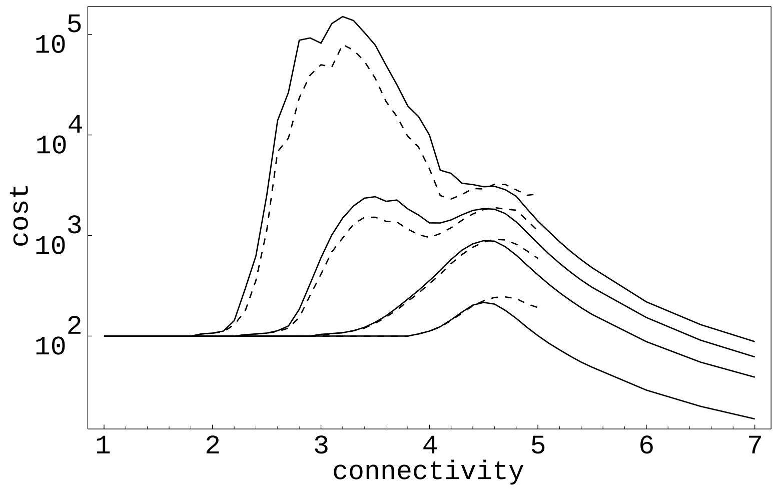


Figure 5.1: Cost percentiles vs. connectivity for 100–node graphs, based on 50000 samples at each value of γ, given in increments of 0.1. Two sets of curves are shown: solid, for the behavior of all samples, and dashed, for those samples with solutions. The dashed curves extend only up to $\gamma = 5$ since beyond that point few instances have solutions. For each set of curves, the lowest shows the 50% cost (i.e., the median). Successively higher curves show the costs for the top 0.05, 0.005 and 0.0005 of the problems.

This behavior also gives rise to a huge variation in search cost at the lower connectivities, which is observed to persist as the size of the problem increases. This is unlike the usual situation in statistical physics where the relative fluctuations in thermodynamic observables go to zero as $1/\sqrt{n}$. Some theoretical understanding of this behavior comes from the variance in the number of solutions for random graphs. This can be calculated by considering the simultaneous pruning of pairs of colorings [62] and differs significantly from the simple result obtained by assuming each solution occurs independently of the others. In particular, when solutions exist but there are not too many, i.e., for intermediate connectivities, the variance in their number is extremely large. Additional variance in the search cost arises from the clustering of solutions in the search space. Nevertheless, this does not completely explain the appearance of the extremely hard cases at intermediate connectivities. These indicate unusual cases in which the pruning is significant only for nearly complete colorings, while being quite low for intermediate-sized partial colorings.

Chapter 6

Problem Ensembles

Most studies of the phase transition have focused on the typical behavior of random problems in which the constraints, i.e., the edges for graph coloring, have no special structure. While simple to generate and treat theoretically, there remains the possibility that ensembles of problems that emphasize more realistic structures or correlations among the constraints will behave in significantly different ways. Although it is difficult to characterize the detailed nature of problems that might generally arise in these cases, there are a number of plausible structural classes as well as some cases from particular applications such as class scheduling [34] or graph colorings that arise in some numerical programs [28].

In this section we examine several such graph ensembles. This shows first, that the transition behavior and existence of an extended distribution is robust in that it appears in these ensembles as well as random graphs, and second, that the choice of ensemble can shift the exact location of the transition point, sometimes significantly. This last observation shows that the location of the transition is somewhat arbitrary in that it depends on the choice of ensemble, and hence is not a direct characterization of topological structure that determines whether an individual problem is critically constrained. These observations may also suggest what additional parameters are needed for accurate statistical models.

To suggest other plausible ensembles, we can consider the origin of various constraint problems and thus the likely structures they will give rise to. In many cases, CSPs correspond to mathematical systems as in a set of axioms used in theorem proving or as a source of theoretically tractable random problems in which there is no special structure or correlations among the components. By contrast, constraints that arise from physical interactions often have a strength that decreases with distance. In discrete systems this gives rise to constraints that involve only variables that are neighbors in some Euclidean lattice structure, such as the nearest-neighbor interactions in the Ising model of spin systems. Similarly, designed artifacts often produce constraints with some locality also. This is because such artifacts are usually nearly decomposable in having strong interactions among a few tightly coupled components and weaker interactions among the rest [52]. Unlike the Euclidean structures of physical systems, the modular construction of many artifacts produces instead hierarchical spaces with an ultrametric topology, i.e., in which interaction strength decreases with distance defined as the number of levels in the hierarchy to the closest common ancestor of two components. This distance metric has the ultrametric property, namely that for any three components we can assign them labels a, b and c, so that $d(a,b) = d(a,c) \geq d(b,c)$. This corresponds to the notion that b and c are in the same module, hence relatively close together, and equally distant from a third component, a, in another module of the artifact. Finally we can consider the constraints that arise in evolved systems, such as biological ecosystems, human market economies and systems of common law, where complex interactions arise through a series of local changes without a central authority's detailed planning. Here too one often observes a clustering of interactions, and this is also suggested theoretically since for stability reasons interactions in large systems must involve relatively few components of the system [36, 37, 20]. Finally, many systems have a mixture of intentional design as well as unplanned incremental evolution as in large software systems and human organizations [44, 18, 4]. These observations suggest that many real-world CSPs will involve more clustering among the constraints than typically found in the random ensemble. It is thus of interest to see whether the phase transition phenomenon persists when there is clustering or other significant deviations from randomness.

6.1 Clustered Graphs

To study the effect of hierarchical clustering, graphs with ultrametric structure were constructed. While there are a number of ways to do this, one simple procedure is to group the nodes of the graph into a binary tree as shown in Fig. 6.1 (which will be unbalanced unless the number of nodes is an exact power of two). This tree induces an ultrametric distance between each pair of nodes, defined as the number of steps up the tree required to reach a common ancestor. A clustered graph was selected by choosing edges between nodes at distance d with relative probability p^d. When $p = 1$ this results in random graphs considered above. For $p < 1$, edges will be more likely between nearby nodes, giving a hierarchical clustering to the graphs.

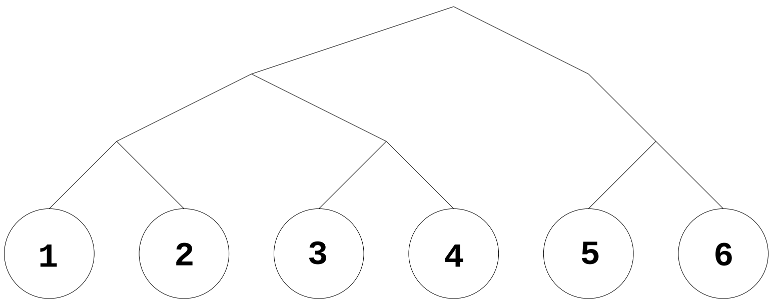


Figure 6.1: Example of an unbalanced binary tree structure for 6 nodes. For a clustered graph, the likelihood of an edge between two nodes decreases with their ultrametric distance, i.e., the number of steps up in the tree to reach a common ancestor. For example, nodes 1 and 2 are at distance 1, while 1 and 5 have distance 3. This tree is used only to define a distance between nodes and should not be confused with the actual constructed graphs, which involve edges between the nodes, represented here as the leaves of the tree.

More specifically, consider the construction of clustered graphs with $n = 2^D$ nodes. In this case the binary tree is balanced and there are 2^{d-1} nodes at distance d from a given one. For each edge to be added, we repeatedly generate pairs of nodes according to the following procedure until a pair is found not already linked by an edge:

1. randomly select a distance d, between 1 and D, inclusive, with relative probability $(2p)^d$
2. select a random node uniformly from the n nodes of the graph
3. select a second random node uniformly from those at distance d from the first

This pair is then linked by an edge.

When n is not an exact power of two, in step 1 we used the smallest value of D such that $n < 2^D$, and the selection of the second node in step 3 was made from among the fully balanced tree. Whenever the selection of the second node gave a node beyond the value of n, that selection was ignored and the procedure repeated. For example, in the case of Fig. 6.1 (where $n = 6$ and $D = 3$), if the distance selected in step 1 were $d = 3$ and the first node selected was node 1, then the second node was selected from among those at distance 3 in the complete tree, i.e., nodes 5 through 8. If the selection actually picked nodes 7 or 8, this selection was ignored and the process repeated. An important point to note is that this procedure could generate *any* graph with n nodes and e edges, just as the random uniform selection does. But they appear with different probabilities. This is in contrast to other forms of clustering in which, say, all edges are restricted to nearby neighbors in which case it would be impossible to ever generate graphs with many edges.

The first consequence of the clustering, shown in Fig. 6.2, is to shift the transition point to lower values of the connectivity, i.e., fewer edges. This is readily understood in that concentrating edges in small groups of nodes makes it more likely there will be no coloring for those nodes, and hence none for the whole graph. Moreover, the typical search costs for these clustered graphs was much larger than for random graphs. This is partly due to the fact that the clustering leads to several relatively large components when there are few edges. By contrast, random graphs tend to have a single giant component containing most of the nodes and a few other tiny components [3] provided $\gamma > 1$. When a graph consists of components with many solutions as well as some without any, it is possible for the Brelaz heuristic described above to repeatedly search for

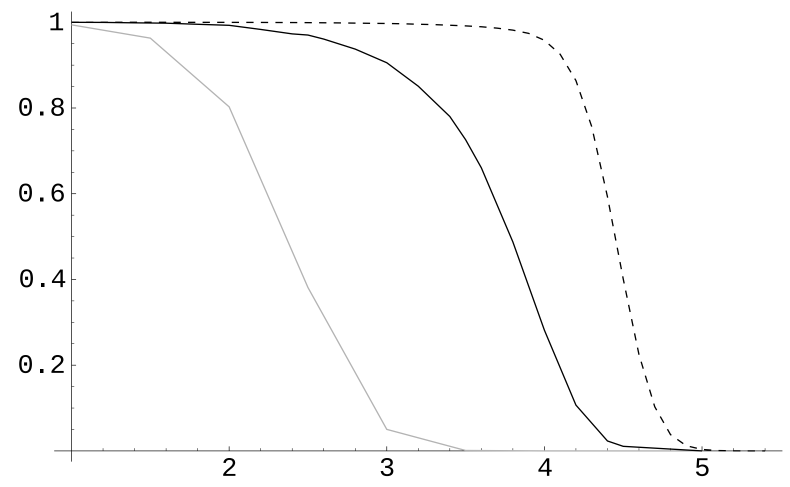


Figure 6.2: Fraction of soluble 100–node clustered graphs vs. connectivity γ. The solid curve is for graphs with $p = 0.5$ with γ in steps of 0.2, while the gray curve is for more tightly clustered graphs with $p = 0.2$ and γ in steps of 0.5. For comparison, the dashed curve is for unclustered random graphs, also shown in Fig. 4.7. We see that more tightly clustered graphs become unsolvable with fewer edges.

all solutions of soluble components, each time failing due to a remaining insoluble component. While this problem occurs only rarely with random graphs, it is significant with clustering. To avoid this, we used a simple modification to the search in which the graph was divided into components and each component searched separately. If any component failed to have a solution, the search was aborted immediately rather than continuing with a fruitless search. This decomposition of the problem into independent pieces can also be viewed as a limited form of dependency-directed backtracking [55], and also suggests that these clustered graphs may benefit from more sophisticated decomposition methods [13].

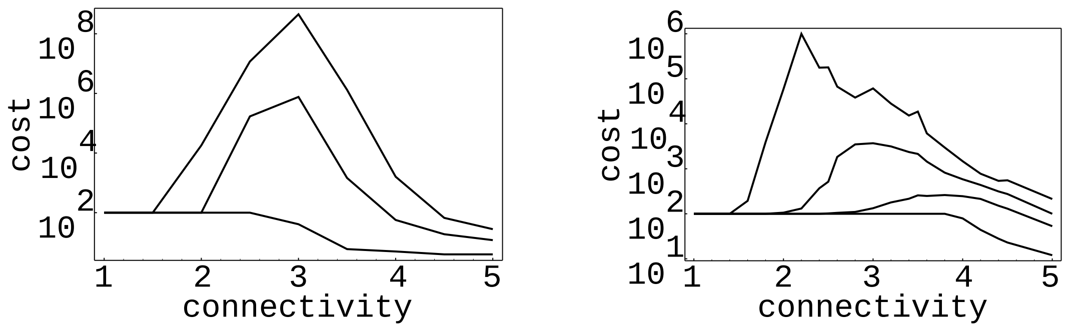


Figure 6.3: Cost percentiles for clustered 100–node graphs using component-based search. The first plot is for $p = 0.2$, showing the 0.5, 0.05 and 0.005 percentiles, based on 1000 samples and γ in steps of 0.5. The second plot is for $p = 0.5$, showing the 0.5, 0.05, 0.005 and 0.0005 percentiles, based on 10000 samples and γ in steps of 0.2.

The search cost for clustered graphs is shown in Fig. 6.3, using the component-based modification to the search method. Compared to the random graphs shown in Fig. 5.1, the higher percentiles are significantly harder, although the median cost is somewhat easier and does not itself display a peak. By contrast, using the component-based search method on random graphs does not appreciably change the results shown in Fig. 5.1. In general we see the same qualitative behaviors: a relatively high density of harder problems near the transition region, as well as occasional hard cases in the underconstrained region below the transition point.

Another form of clustering is due to problems that arise from physical interactions. These are usually localized in space so their representation in discrete constraint systems will usually involve many constraints between components that are physically close in the system, while those far from each other have little or no interaction. Thus an interesting follow-up to the above observations on hierarchical clustering would be to study clustering based on an embedding in physical space. In the context of graph coloring, this could be achieved by placing the nodes on a grid of two or three dimensions and favoring edges between nodes that are close to each other on the grid according to the usual euclidean metric. While such lattice structures are commonly studied as models of physical systems, they have not yet been used as models of constraint satisfaction problems to investigate the extent of the phase transition phenomenon in search.

6.2 Connected Graphs

Another common characteristic of actual search problems is that many "trivial" aspects of the problem would normally be removed before resorting to a lengthy search method. One immediate example is that problems that decompose into completely separate subproblems can have each subproblem solved independently. This is particularly important for simple search methods that do not themselves exploit the problem's decomposability. The Brelaz heuristic search used here is an intermediate case in that its nodes are selected from one component at a time so there will be no backtracking through previously colored components for cases in which a solution exists. However, when a solution does not exist, there is the potential for wasted repeated searches of previously colored components. This raises the question of whether the behaviors seen for randomly generated problems are significantly different for connected graphs, i.e., the type of problem that remains after decomposition into independent parts.

It is conceptually straightforward to generate graphs uniformly from the set of connected graphs with a given number of edges [56], but this is computationally demanding. A simpler approach, as used here, is to first construct a random tree [41] on the n nodes of the graph, which uses $n-1$ edges, and then add the remaining edges randomly. This is guaranteed to produce connected graphs, but they will not be uniformly selected. In fact, there is no particular reason to suppose that the connected components that arise in realistic problems correspond to uniform selection among connected graphs, thus motivating a choice based on computational simplicity. This method of generating connected graphs has also been used in other studies of constraint satisfaction [13]. Note that connected graphs require that $e \geq n-1$ or $\gamma \geq 2(1-1/n)$.

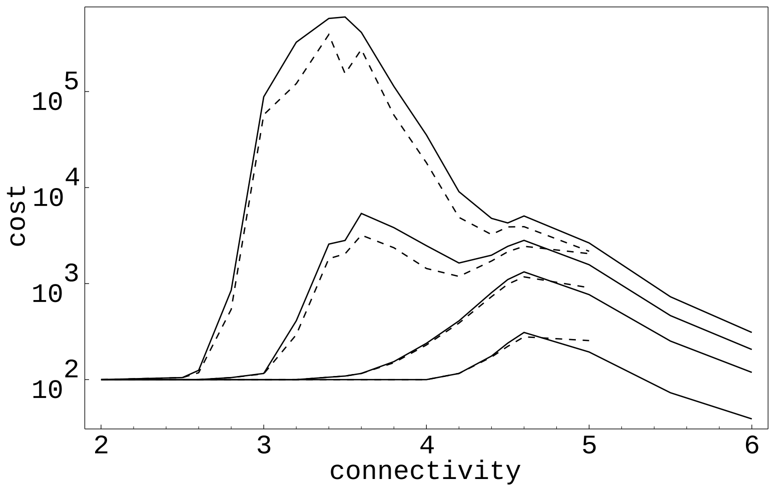


Figure 6.4: Cost percentiles vs. connectivity for 100–node connected graphs, sampled as described in the text. These are based on 50000 samples at each value of γ, given in increments of 0.2. The solid curves include all samples and the dashed curves are for those with a solution. The curves, from bottom to top, correspond to the 0.5, 0.05, 0.005 and 0.0005 percentiles.

The resulting behavior, shown in Fig. 6.4, is very similar to the behavior of random graphs in having a transition from under- to overconstrained problems as well as producing exceptionally hard problems below this transition point. Quantitatively, this has somewhat higher cost for intermediate connectivities, e.g., between 3 and 4, and lower cost for small connectivities, where the graphs are trees with just a few additional edges. However, those rare cases near $\gamma = 2$ that had no solution were found to have extremely high search cost. In addition, the transition occurs near $\gamma = 4.8$, slightly above the location for random graphs.

6.3 Padded Graphs

Preprocessing is another technique for removing trivial parts of a constraint problem before starting the search. In the case of graph coloring using b colors, nodes in a graph with fewer than b neighbors can always be colored, no matter what colors are assigned to the neighbors. Thus a useful preprocessing method is to remove any such nodes from the graph, repeating this procedure until all remaining nodes have at least b neighbors. To examine the effect of this type of preprocessing, we can consider the behavior of search applied to graphs in which each node is guaranteed to have at least b neighbors. These graphs were constructed by examining each node in turn and randomly adding enough edges to it to produce at least b neighbors. Note that this requires $\gamma \geq b$ and may not complete successfully, especially when γ is close to this lower limit. In these cases, the construction was repeated until each node was successfully given at least b neighbors. A more efficient alternative in these cases would be to start with a random regular graph [56] in which each node has exactly b neighbors. Remaining edges were then added randomly. We denote this as the class of "padded graphs".

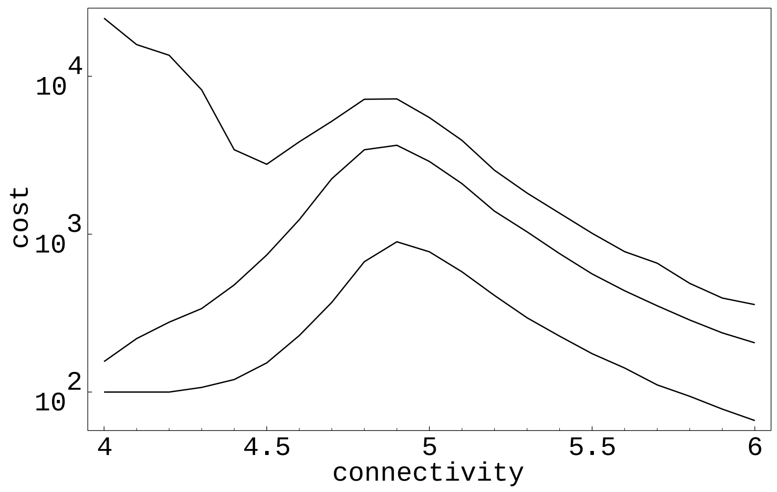


Figure 6.5: Cost percentiles vs. connectivity for 100–node padded graphs, based on 10000 samples at each value of γ in increments of 0.1. The curves, from bottom to top, correspond to the 0.5, 0.05 and 0.005 percentiles.

The resulting search cost distribution is shown in Fig. 6.5. These graphs are generally harder than uniform random ones, and the transition from under- to overconstrained cases occurs at $\gamma = 4.9$, slightly higher than for random graphs. The transition behavior is also seen with more elaborate preprocessing operations [6].

6.4 Graphs with Prespecified Coloring

In many studies of constraint satisfaction, problems that are known to have a solution are used. These are particularly useful for studies of incomplete search methods, such as heuristic repair or simulated annealing, that would never terminate on a problem with no solutions. Such a class of graphs could be obtained by generating random problems and keeping only those that have a solution. However, this becomes very inefficient in the overconstrained region since there almost all random instances will have no solutions. Instead, we prespecify a solution and avoid any constraints that would exclude this solution. Specifically, for graph coloring with b colors, we initially divide the nodes into b groups, as nearly equally as possible. Edges are then added randomly, except any that would link nodes in the same group are not allowed. Thus it will always be possible to find a coloring for the resulting graphs, namely by assigning the same color to each node in a group, and distinct colors to the groups. While this construction is biased toward cases with many solutions and so results in relatively easier graphs, it is efficient and commonly used as the class of soluble problems both in theoretical and empirical studies.

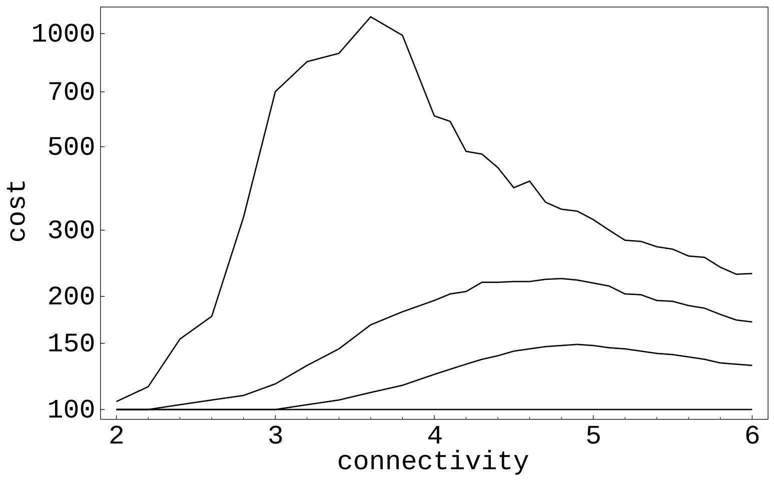


Figure 6.6: Cost percentiles vs. connectivity for 100–node soluble graphs, based on 50000 samples at each value of γ in increments of 0.1. The curves, from bottom to top, correspond to the 0.5, 0.05, 0.005 and 0.0005 percentiles. Note that the 0.5 percentile (the median) is 100 for all connectivities. A peak in the median is seen only for larger graphs.

The resulting cost distribution is shown in Fig. 6.6. Note that this class of graphs is significantly easier than the other classes. In particular, the median cost remains at 100 for all connectivities. However, the median does show a peak when larger graphs are considered, e.g., with 150 nodes. For this class of graphs, there is no transition in the fraction of soluble graphs (which is always one), but the more general notion of a transition from under- to overconstrained problems still applies [62] and corresponds to the peak in the 0.05 percentile, or in the median cost for larger graphs. Specifically, additional edges now prune only those partial and complete solutions that do not correspond to the prespecified grouping of the nodes. In overconstrained problems, almost all such solutions are pruned and the search rapidly discovers the prespecified grouping. In this case, the transition takes place at $\gamma = 4.8$. As with the other classes of graphs, we see a shift toward lower connectivities for the peak in the higher percentiles.

Chapter 7

Refining the Transition

The large variance and dependence of the transition on the specific definition of the class of problems considered suggest that the parameters used to specify the problem and search are not sufficiently precise. To have confidence that most observed searches will be fairly close to the expected theoretical behavior, a more detailed description of the search problems is required. Transitions in a class of problems arise when there is a high density of hard cases. This depends not only on the existence of hard instances, but also on their likelihood in a particular class of problems. Since this latter characteristic is defined by the construction of the class rather than the behavior of individual instances, it is not surprising that hard cases need not be confined to just the transition region of a problem class.

Fundamentally, the hard cases appear to be associated with "critically constrained" problems. Such problems are characterized by having many partial solutions of large size (i.e., consistent colorings of large portions of the graph) but very few complete solutions. Thus one cause of the variation is that the given parameters to not tightly determine the number of partial and complete solutions for a class of problems.

In addition to the *number* of solutions, their clustering also contributes to the variance in the search cost. Specifically, with a fixed number of solutions, additional clustering means that poor initial choices can lead to searching very many unproductive partial colorings [53]. Thus even if the number of solutions were accurately known, there could still be significant variation in search cost due to differences in how tightly clustered the solutions are. It thus remains an open question whether additional parameters will be required to specify the clustering sufficiently tightly even if the number of solutions were well specified. Note that the amount of clustering of the *solutions* is distinct from the clustering of the *constraints*: even uniform random graphs have significant solution clustering. This is due to the simple observation that a solution, which satisfies all the constraints, is much more likely to have neighboring states (i.e., states that differ by only a few color assignments) that also satisfy all the constraints than is the case for a randomly selected initial coloring.

A better specification of the number and location of solutions requires more information about the structure of the problems, but is independent of the search method used. However, search methods themselves can differ in how well they avoid unproductive choices, e.g., by their use of heuristics to select new variables or values to consider, where to backtrack to as well as any additional domain-specific information that may be available in particular cases. Such differences can lead to additional variation in observed search cost.

In this section, we present additional structural parameters that better characterize the number of solutions and heuristic effectiveness of a search method.

7.1 Characterizing Problem Structure

While Eq. (4.1) gives the expected number of partial colorings, there is a large variation among graphs with a given number of edges. One way to make more precise predictions of these values and hence obtain a more sensitive measure of problem difficulty, is to use additional information about the graph.

We can proceed in a systematic manner to obtain additional structural parameters. Consider a given coloring problem with b colors, n nodes and e edges. Let E_k be the set of colorings for the graph that are eliminated by edge k (namely, all those colorings for which the nodes connected by that edge have the same color). We define the size of the intersections of these sets

$$S_r = \sum |E_{l_1} \cap \ldots \cap E_{l_r}| \tag{7.1}$$

where the sum is over all r-subsets of $\{1, \ldots, e\}$, and we define $S_0 = b^n$, the total number of possible colorings. Then the principle of inclusion-exclusion [42] gives the number of colorings that are not in any of the E_k, i.e., the number of solutions:

$$N = \sum_{i=0}^{e} (-1)^i S_i \tag{7.2}$$

Each edge eliminates b^{n-1} colorings, so $S_1 = \sum_{l=1}^{e} b^{n-1} = eb^{n-1}$. The colorings that are eliminated by both of two edges require that two nodes have the same color and that either another pair have the same color or a third node has the same color as the first two. In both cases we have b^{n-2} eliminated colorings, giving $S_2 = \binom{e}{2} b^{n-2}$. Thus we see that the first terms in the inclusion-exclusion expansion for the number of solutions depend only on the number of edges in the graph. With three edges, we can have from three to six nodes involved. When there are only three nodes (a triangle), the third edge does not give any additional pruning, i.e., there are b^{n-2} colorings pruned by all three edges. Otherwise, each edges adds a further restriction pruning only b^{n-3}. Thus we have $S_3 = \binom{e}{3} b^{n-3} + (b^{n-2} - b^{n-3})t$ where t is the number of triangles in the graph. Further terms in Eq. (7.2) involve more complex subgraphs, e.g., squares.

This can also be directly related to Eq. (4.1) by noting that with r edges we have b^{n-r} states pruned by all r edges when there are no "redundant" nodes linked by the edges (i.e., nodes that are already constrained by other edges). Ignoring such cases, i.e., assuming each edge acts independently, gives $S_r \approx \binom{e}{r} b^{n-r}$ and hence

$$N \approx b^n \left(1 - \frac{1}{b}\right)^e \tag{7.3}$$

reproducing Eq. (4.1) for N_n.

To see the effect of including triangles, consider the probability that a given complete coloring is consistent with three edges. If these edges do not form a triangle, then they constrain three nodes to only have $b-1$ color choices. Hence the state will satisfy the three constraints with probability $(b-1)^3/b^3$, which is equivalent to the result obtained by assuming the edges act independently. If, however, the three edges form a triangle, then the number of acceptable colorings is $b(b-1)(b-2)$ so the probability the state satisfies the three constraints is $(b-1)(b-2)/b^2$, which is somewhat less than assuming independence. Thus we see that triangles reduce the number of solutions below what one would expect by assuming independence.

This suggests that the number of triangles in a graph gives a finer determination of classes of critically constrained graphs than just knowing the number of edges. That this is in fact the case is shown in Fig. 7.1. In particular, we see that there is a transition from mostly solvable to mostly unsolvable cases depending on both the connectivity and the number of triangles. This also shows that a transition from under- to overconstrained problems occurs for a range of connectivity values, providing an explanation for the hard search cases for the lower connectivities as seen in Fig. 5.1.

A remaining question is how well this parameter discriminates among hard and easy problems. To address this question, a large number of uniformly generated random graphs were searched and the results grouped according to how many triangles each graph had. This is a simple procedure to generate graphs with given connectivity and number of triangles uniformly, but is inefficient for producing many samples of graphs whose number of triangles differs greatly from the average value for random graphs at a given connectivity. The results are shown in Fig. 7.2 for two different values of the connectivity. The plots are qualitatively similar to the behavior shown in Fig. 5.1, i.e., there is an easy-hard-easy pattern and the higher percentiles show a shift toward less constrained region. So, using the number of triangles as an additional parameter more precisely specifies the transition from under- to overconstrained problems, but the variance remains large and some hard cases continue to occur below the transition point.

7.2 Characterizing Search Methods

A further reason for high variance is the search method itself. To characterize the search heuristic, we return to the simple theory of Eq. (4.1). A search is difficult when many partial colorings that do not lead to

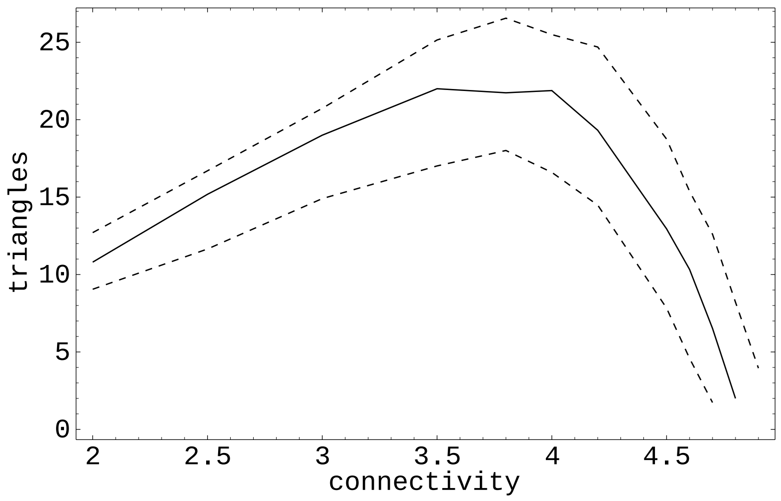


Figure 7.1: Fraction of cases with a solution as a function of number of triangles in the graph and γ for $n = 100$, $b = 3$. Values are shown for γ in increments of 0.5 except for increments of 0.1 beyond $\gamma = 4.5$. The solid curve shows where half the graphs have a solution, while the upper and lower dashed curves show where fractions 0.2 and 0.8 have solutions, respectively.

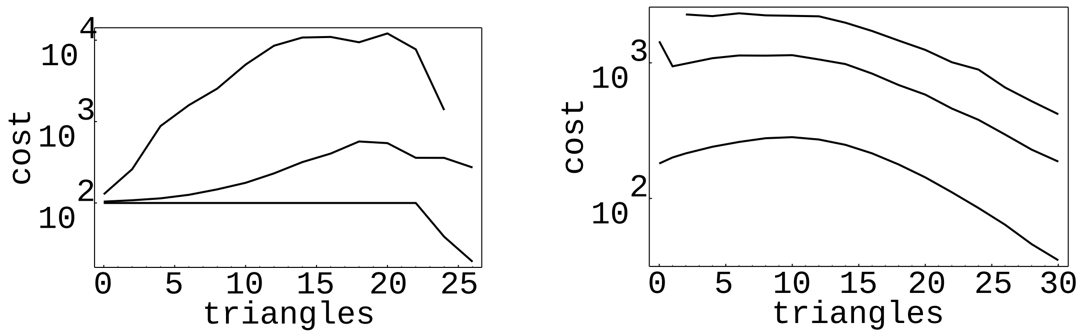


Figure 7.2: Cost distributions vs. number of triangles for $n = 100$, $b = 3$ and $\gamma = 3.5$ (on the left) and 4.5. Shown are the percentiles for the top 0.5, 0.05 and 0.005 of the problems. Generally, 10000 samples were obtained at each number of triangles, in increments of 2. However, only a few hundred samples were obtained for the cases of $\gamma = 3.5$ with more than 20 triangles.

solutions must be considered. A measure of the effectiveness of a heuristic method, i.e., its ability to prune unproductive nodes, can be simply quantified as the probability p that an unproductive partial coloring will be recognized as such in the search. In practice, such a parameter can be estimated by sampling [7]. A partial coloring of size k is considered only if the smaller colorings preceding it in the search order did not eliminate it, i.e., with probability $(1-p)^{k-1}$ if the pruning ability is independent for each sequence of possible colorings.

The result of this pruning is most simply illustrated when there is no solution. In such a case the expected overall search cost is $C = N_0 + \sum_{k=1}^{n}(1-p)^{k-1}N_k$, whose behavior is shown in Fig. 7.3. Note first that the cost decreases rapidly as the heuristic is improved. A more subtle observation is the change in behavior as the problem size increases: for poor heuristics, i.e., small p, the cost grows exponentially fast whereas for large p, the cost is roughly independent of problem size.

This behavior is conceptually straightforward. If the heuristic can prune unproductive colorings faster than their number grows with size k, the search will involve fairly limited backtracking giving a low cost. This will be the case when $p > 1/b$ so on average each unproductive partial coloring of size k produces less than one additional coloring, of size $k+1$, to include in the search. A less powerful heuristic results in a search involving considerable backtracking and an exponentially growing cost. We thus see another

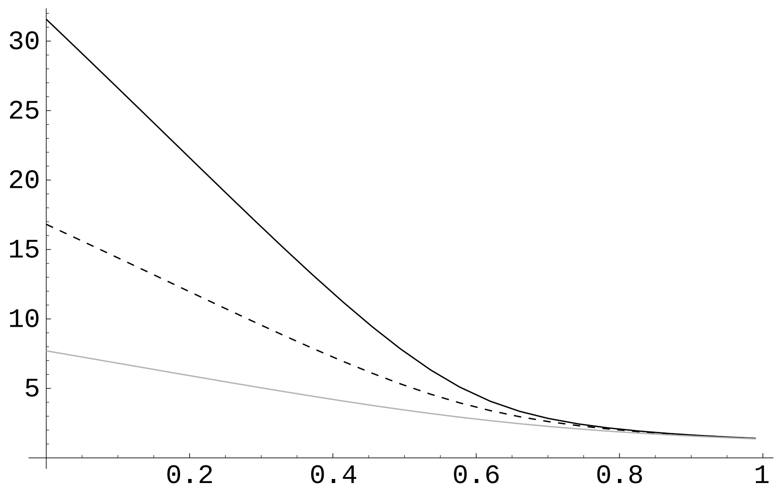


Figure 7.3: Behavior of search cost when there are no solutions as a function of heuristic pruning effectiveness. The plot shows $\ln C$ vs. p for n of 20 (gray), 50 (dashed) and 100 (solid), with $\gamma = 6$ so there are unlikely to be any solutions.

transition behavior, this time due to changes in search method rather than the underlying structure of the problem. This transition behavior can also be found in more elaborate models, e.g., including the possibility that the heuristic incorrectly prunes a partial coloring that does lead to a solution (in which case there is the possibility that the search will incorrectly conclude there are no solutions when in fact there are) as well as cases where there is a solution [26]. More generally, this illustrates that heuristics can allow the pruning potential of the search and the structure of the problem, as characterized by the number of solutions, to vary separately [61].

Chapter 8

Deceptive Hard Problems

An appealing result of the studies of this phase transition is that the peak in median search cost, for a variety of search methods, is associated with an algorithm-independent property of the class of problems, namely a transition from under- to overconstrained instances. This suggests that the structural parameters are, at least partly, characterizing "intrinsically hard" problems rather than just those that are difficult for specific heuristic search methods.

One commonly used qualitative explanation for the intrinsically hard cases is that they are due to the existence of many large partial solutions but few, if any, complete ones. A complementary view is that the hard cases are "critically constrained", that is they have just enough constraints to eliminate most or all of the complete colorings but not enough constraints to allow significant pruning early in the search. This latter view derives from the observation, seen in Fig. 4.9, that constraints are much more likely to prune large states than small ones.

In this section we investigate these claims with a simple example, showing that although they are necessary for a problem to be intrinsically hard, they are not, as they stand, sufficient. We focus on an insoluble example, but note that such cases should also be relevant for hard soluble instances as well. This is because a hard soluble instance is hard precisely because the first few search choices often produce a hard, insoluble subproblem which must then be searched completely before backtracking far enough back to correct these early choices [16, 53].

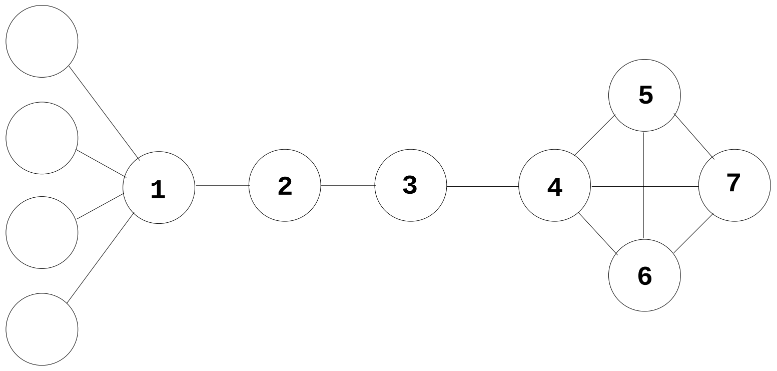


Figure 8.1: A difficult graph structure for the Brelaz heuristic to 3–color. This example has 11 nodes. The numbers show the order in which the Brelaz heuristic colors the nodes (except that nodes 5 and 6 can be selected in either order). The unnumbered nodes are never colored during the search. Larger versions of this structure are created by adding additional nodes in a chain between nodes numbered "2" and "3". Such graphs are easy for more sophisticated search methods such as dependency-directed backtrack or using a preprocessing step that eliminates nodes with fewer than three neighbors.

An example of a type of graph that will "fool" the Brelaz heuristic is shown in Fig. 8.1. Such graphs consist of a "star" of degree d connected by a long chain of nodes to the complete graph with four nodes, K_4. For the example in the figure, $d = 5$ with the node numbered "1" forming the center of the star. This graph has $e = n + 2$ edges so $\gamma = 2 + 4/n$. Because the Brelaz heuristic always starts with the node of highest degree in the graph, when $d \geq 5$ the center of the star will always be the first node colored. Successive choices will assign colors to nodes along the chain, in order, since the heuristic selects nodes with the fewest remaining color choices, and among those, nodes with the most uncolored neighbors. Eventually, the search reaches the last node of K_4, and finds no additional choices. This forces extensive backtracking, and high search cost.

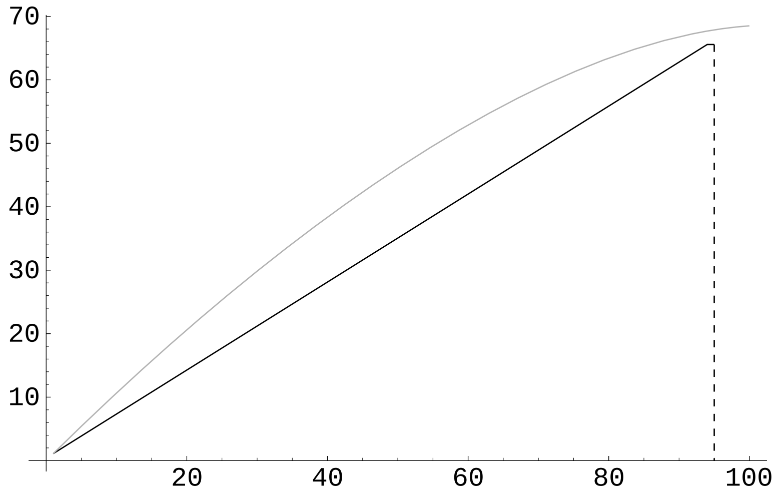


Figure 8.2: Number of partial colorings vs. their size k for a deceptively hard node graph with $n = 100$ and $d = 5$. Specifically, the black curve shows $\ln n_k$, with the dashed step indicating the point at which n_k abruptly becomes zero. For comparison, the gray curve shows $\ln N_k$, given in Eq. (4.1), for the corresponding case with $e = n+2$ edges.

The resulting number of partial colorings of size k, in the order examined by this heuristic, is given by

$$n_k = \begin{cases} 3\ 2^{k-1} & 1 \leq k \leq n-d-1 \\ 3\ 2^{n-d-2} & k = n-d \\ 0 & n-d < k \leq n \end{cases} \tag{8.1}$$

because the first node has 3 color choices, subsequent ones along the chain each have 2 possibilities, and the last two nodes within the K_4 part of the graph have 1 and 0 possibilities, respectively. This is shown in Fig. 8.2, and compared with the expected behavior for random graphs with the same number of edges.

Thus we see that this graph, using the Brelaz heuristic, is hard precisely because there are many large partial solutions but no complete solutions. This is in fact true of most other orderings of the nodes: pruning occurs only when all of K_4 is finally considered, and relatively few of the possible orderings place these nodes near the start of the search. However, this is not an example of an "intrinsically hard" graph since more sophisticated search methods can deal with it easily. Moreover, removing any single edge from the K_4 part of the graph changes it into a trivial, backtrack-free search problem for the Brelaz heuristic.

The lesson from this simple example is that intrinsically hard problems must have a structure that gives rise to many large partial solutions no matter which ordering is selected (the other aspect of this requirement for hardness, i.e., also having few or no complete solutions, does not depend on the ordering chosen). Similarly, the notion of "critically constrained" must be more precise in requiring that in fact most of the constraints are needed to eliminate the potential solutions. Topologically, this would correspond to an intricate combination of many constraints that manage to collectively prune the complete states, without allowing much pruning from any small subset of the constraints.

Thus an important open issue concerns identifying local, easily computed structural properties of the graphs that can readily distinguish when a problem is likely to be intrinsically hard or just deceptively hard for a particular search method, in which case a more powerful method might be able to solve the problem with relatively little search. Note that this is not a simple choice in that there can be cases in which preprocessing actually interferes with the more sophisticated search strategies [46]. Thus, in general, we can expect that within a class of problems, each search method will have its own set of hard cases, some of which will be relatively easy for other methods while others will be difficult for all. An interesting question then is how, among the hard problems for a given search method, the ratio of deceptively to intrinsically hard problems varies among different classes of problems. This also has implications for generating hard problem examples for testing combinations of search algorithms [24, 11], namely at least checking that the presumed hard examples are indeed hard with respect to a variety of search methods.

Chapter 9

Discussion

In summary, we gave a description of the behavior of combinatorial search using graph coloring as an illustration. This shows a phase transition in problem solubility and a corresponding high density of relatively hard instances. Unfortunately, the large remaining variance and dependence on the specific class of problems prevents accurate predictions of individual cases. To partially address this difficulty, we introduced an additional structural parameter which refines the location of critically constrained problem instances. We also described a characterization of heuristic search pruning. These additions give a more precise characterization of the behavior and can be useful for discriminating among the likely behaviors of search problems as well as constructing hard cases.

There remain several open issues. These include developing a simple theoretical understanding of the combination of these parameters, analogous to that given by Eq. (4.1), and understanding the remaining variation which may be partially due to solution clustering. At a more formal level, we lack a definitive theoretical analysis to establish the asymptotic existence and location of the transition as problem sizes increases, in much the same way as other transition behaviors in random graphs have been proven [3]. Nevertheless, in practice, the simple theory, based on independence assumptions, already provides a qualitative understanding of the transition behavior, and in many cases gives reasonable quantitative accuracy [47, 60]. Moreover, because of the dependence of the transition on the exact choice of problem ensemble, a complete analysis of one ensemble, e.g., uniformly selected random graphs, would be of limited applicability.

There is also the question of characterizing the topological structure underlying the hard, critically-constrained problem instances. The concentration of hard cases in the transition region provides some clues as to this structure, and can usefully be applied to generate hard instances. However, knowledge of these transition points does not definitively characterize the structure of hard instances because the transition location depends on the choice of problem class. Conversely, individual problem instances belong to many possible classes (with different likelihoods of appearing). Moreover, hard problems occur outside the transition region, in particular, in regions that are less constrained on average, and conversely the transition regions contain many easy problems as well.

On a more general note, if these behaviors applied only to coloring random graphs, or only to the particular heuristic search used here, they would be of limited interest. However, these transitions have been commonly reported for a variety of combinatorial search problems with a range of very different search methods, as mentioned in the introduction. Such complexity transitions arise not only for constraint satisfaction problems, such as graph coloring, but also for other cases such as optimization [64].

The additional parameter for problem structure generalizes to other constraint problems based on the inclusion-exclusion result for the number of solutions. These problems can all be viewed as due to local inconsistencies from the constraints combining to determine which complete states satisfy all the constraints [62]. In this case the additional parameter relates to the degree of overlap among the local inconsistencies. More accurate theoretical values for the transition points are possible [60] by including some account of the dependencies among the constraints.

Finally, there are applications of these results to the design of better search algorithms. For example, the theory gives a domain-independent heuristic to identify subproblems that are likely to be particularly hard or easy, on average. This can be used to improve genetic algorithms [8] and may also suggest better choices for backtracking. Knowledge of the likely difficulty of search problems can also be used to determine when a diverse set of methods, perhaps running in parallel and sharing partial results, is most useful [24, 11]. In these cases, improved understanding of the location and nature of the phase transitions in combinatorial search problems can translate directly into more effective search methods.

9.1 Acknowledgments

M. Lomelin, A. Sahai and C. Williams contributed to many helpful discussions and the development of the search programs used to produce these results.

Bibliography

[1] Andrew B. Baker. The hazards of fancy backtracking. In *Proc. of the 12th Natl. Conf. on Artificial Intelligence (AAAI94)*, pages 288–293, Menlo Park, CA, 1994. AAAI Press.

[2] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1973.

[3] B. Bollobas. *Random Graphs.* Academic Press, NY, 1985.

[4] R. S. Burt. Models of network structure. *Annual Review of Sociology*, 6:79–141, 1980.

[5] Tom Bylander. An average case analysis of planning. In *Proc. of the 11th Natl. Conf. on Artificial Intelligence (AAAI93)*, pages 480–485, Menlo Park, CA, 1993. AAAI Press.

[6] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In J. Mylopoulos and R. Reiter, editors, *Proceedings of IJCAI91*, pages 331–337, San Mateo, CA, 1991. Morgan Kaufmann.

[7] Pang C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal of Computing*, 21(2):295–315, April 1992.

[8] Scott H. Clearwater and Tad Hogg. Exploiting problem structure in genetic algorithms. In *Proc. of the 12th Natl. Conf. on Artificial Intelligence (AAAI94)*, pages 1310–1315, Menlo Park, CA, 1994. AAAI Press.

[9] James M. Crawford and Larry D. Auton. Experimental results on the cross-over point in satisfiability problems. In *Proc. of the 11th Natl. Conf. on Artificial Intelligence (AAAI93)*, pages 21–27, Menlo Park, CA, 1993. AAAI Press.

[10] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[11] Pedro S. de Souza and Saroush Talukdar. Asynchronous organizations for multi-algorithm problems. In *Proc. of ACM Symposium on Applied Computing (SAC93)*, pages 286–294, Feb. 1993.

[12] M. E. Dyer and A. M. Frieze. The solution of some random NP-hard problems in polynomial expected time. *J. of Algorithms*, 10:451–489, 1989.

[13] Eugene C. Freuder and Paul D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proc. of IJCAI93*, pages 254–260, San Mateo, CA, 1993. Morgan Kaufmann.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[15] Ian P. Gent and Toby Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.

[16] Ian P. Gent and Toby Walsh. The hardest random SAT problems. Technical Report 680, Dept. of AI, Univ. of Edinburgh, January 1994.

[17] Craig Gotsman. A cluster detection algorithm based on percolation theory. *Pattern Recognition Letters*, 12:199–202, April 1991.

[18] J. A. Hartigan. Representation of similarity matrices by trees. *Journal of the American Statistical Association*, 62:1140–1158, 1967.

[19] T. Hogg and B. A. Huberman. Artificial intelligence and large scale computation: A physics perspective. *Physics Reports*, 156:227–310, 1987.

[20] T. Hogg, B. A. Huberman, and Jacqueline M. McGlade. The stability of ecosystems. *Proc. of the Royal Society of London*, B237:43–51, 1989.

[21] Tad Hogg. Phase transitions in constraint satisfaction search. A World Wide Web page with URL http://www.parc.xerox.com/dynamics/www/constraints.html, 1994.

[22] Tad Hogg. Statistical mechanics of combinatorial search. In *Proc. of the Workshop on Physics and Computation (PhysComp94)*, pages 196–202, Los Alamitos, CA, 1994. IEEE Press.

[23] Tad Hogg and J. O. Kephart. Phase transitions in high-dimensional pattern classification. *Computer Systems Science and Engineering*, 5(4):223–232, October 1990.

[24] Tad Hogg and Colin P. Williams. Solving the really hard problems with cooperative search. In *Proc. of the 11th Natl. Conf. on Artificial Intelligence (AAAI93)*, pages 231–236, Menlo Park, CA, 1993. AAAI Press.

[25] Tad Hogg and Colin P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69:359–377, 1994.

[26] B. A. Huberman and T. Hogg. Phase transitions in artificial intelligence systems. *Artificial Intelligence*, 33:155–171, 1987.

[27] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May-June 1991.

[28] Mark T. Jones and Paul E. Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14(3):654–669, 1993.

[29] P. Kanerva. Self-propagating search: A unified theory of memory. Technical Report CSLI-84-7, Stanford Univ., 1984.

[30] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[31] Scott Kirkpatrick and Bart Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.

[32] Tracy Larrabee and Yumi Tsuji. Evidence for a satisfiability threshold for random 3CNF formulas. In Haym Hirsh et al., editors, *AAAI Spring Symposium on AI and NP-Hard Problems*, pages 112–118. AAAI, 1993.

[33] Manhot Lau and Takashi Okagaki. Applications of the phase transition theory in visual recognition and classifications. *J. of Visual Communication and Image Representation*, 5(1):88–94, 1994.

[34] Gary Lewandowski and Anne Condon. Experiments with parallel graph coloring heuristics. In *Proc. of 2nd DIMACS Challenge*, 1993.

[35] A. K. Mackworth. Constraint satisfaction. In S. Shapiro and D. Eckroth, editors, *Encyclopedia of A.I.*, pages 205–211. John Wiley and Sons, 1987.

[36] R. M. May. Will a large complex system be stable? *Nature*, 238:413–414, 1972.

[37] Ross E. McMurtrie. Determinants of stability of large randomly connected systems. *J. Theor. Biol.*, 50:1–11, 1975.

[38] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[39] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proc. of the 10th Natl. Conf. on Artificial Intelligence (AAAI92)*, pages 459–465, Menlo Park, 1992. AAAI Press.

[40] Paul Morris. On the density of solutions in equilibrium points for the queens problem. In *Proc. of the 10th Natl. Conf. on Artificial Intelligence (AAAI92)*, pages 428–433, Menlo Park, CA, July 1992. AAAI Press.

[41] A. Nijenhuis and H. S. Wilf. *Combinatorial Algorithms for Computers and Calculators.* Academic Press, New York, 2nd edition, 1978.

[42] E. M. Palmer. *Graphical Evolution: An Introduction to the Theory of Random Graphs.* Wiley Interscience, NY, 1985.

[43] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, Mass, 1984.

[44] Alan J. Perlis. Another view of software. In *Proceedings of the 8th International Conference on Software Engineering (ICSE85)*, pages 395–396. ACM, 1985.

[45] Patrick Prosser. Bm+bj=bmj. In *Proc. of the 9th Conf. on AI for Applications*, pages 257–262. IEEE Press, 1993.

[46] Patrick Prosser. Domain filtering can degrade intelligent backtracking search. In *Proc. of IJCAI93*, pages 262–267, San Mateo, CA, 1993. AAAI, Morgan Kaufmann.

[47] Patrick Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. Technical Report AISL-49-93, Dept. of Computer Science, Univ. of Strathclyde, Glasgow G1 1XH, Scotland, 1993.

[48] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[49] Bart Selman and Henry Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc. of IJCAI93*, pages 290–295, San Mateo, CA, 1993. Morgan Kaufmann.

[50] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the 10th Natl. Conf. on Artificial Intelligence (AAAI92)*, pages 440–446, Menlo Park, CA, 1992. AAAI Press.

[51] J. Shrager, T. Hogg, and B. A. Huberman. Observation of phase transitions in spreading activation networks. *Science*, 236:1092–1094, 1987.

[52] H. Simon. *The Sciences of the Artificial.* M.I.T. Press, Cambridge, MA, 1969.

[53] Barbara M. Smith. In search of exceptionally difficult constraint satisfaction problems. In *Proc. of ECAI94 Workshop on Constraint Processing*, pages 79–86, 1994.

[54] Barbara M. Smith. The phase transition in constraint satisfaction problems: A closer look at the mushy region. Technical Report 93.41, Division of AI, Univ. of Leeds, Leeds LS2 9JT, U.K., 1994.

[55] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artifical Intelligence*, 9:135–196, 1977.

[56] G. Tinhofer. On the generation of random graphs with given properties and known distributions. *Appl. Comput. Sci., Ber. Prakt. Inf.*, 13:265–297, 1979.

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

[58] M. S. Waterman, L. Gordon, and R. Arratia. Phase transitions in sequence matches and nucleic acid structure. *Proc. Natl. Acad. Sci. USA*, 84:1239–1243, 1987.

[59] Colin P. Williams and Tad Hogg. Using deep structure to locate hard problems. In *Proc. of the 10th Natl. Conf. on Artificial Intelligence (AAAI92)*, pages 472–477, Menlo Park, CA, 1992. AAAI Press.

[60] Colin P. Williams and Tad Hogg. Extending deep structure. In *Proc. of the 11th Natl. Conf. on Artificial Intelligence (AAAI93)*, pages 152–157, Menlo Park, CA, 1993. AAAI Press.

[61] Colin P. Williams and Tad Hogg. The typicality of phase transitions in search. *Computational Intelligence*, 9(3):221–238, 1993.

[62] Colin P. Williams and Tad Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[63] D. C. Wood. A technique for coloring a graph applicable to large-scale optimization problems. *Computer Journal*, 12:317, 1969.

[64] Weixiong Zhang and Richard E. Korf. An average-case analysis of branch-and-bound with applications: Summary of results. In *Proc. of the 10th Natl. Conf. on Artificial Intelligence (AAAI92)*, pages 545–550, Menlo Park, CA, 1992. AAAI Press.

[65] Weixiong Zhang and Joseph C. Pemberton. Epsilon-transformation: Exploiting phase transitions to solve combinatorial optimization problems - initial results. In *Proc. of the 12th Natl. Conf. on Artificial Intelligence (AAAI94)*, pages 895–900, Menlo Park, CA, 1994. AAAI Press.