Computational Intelligence, Volume ?, Number ?, 1998

ANALYSIS OF HEURISTIC FOR NUMBER PARTITIONING

IAN P. GENT AND TOBY WALSH
{*ipg,tw*}*@cs.strath.ac.uk*
Department of Computer Science,
University of Strathclyde,
Glasgow G1 1XH,
Scotland.

We illustrate the use of phase transition behaviour in the study of heuristics. Using an "annealed" theory, we define a parameter that measures the "constrainedness" of an ensemble of number partitioning problems. We identify a phase transition at a critical value of constrainedness. We then show that constrainedness can be used to analyse and compare algorithms and heuristics for number partitioning in a precise and quantitative manner. For example, we demonstrate that on uniform random problems both the Karmarkar-Karp and greedy heuristics minimize the constrainedness, but that the decisions made by the Karmarkar-Karp heristic are superior at reducing constrainedness. This supports the better performance observed experimentally for the Karmarkar-Karp heuristic. Our results refute a conjecture of Y. Fu that phase transition behaviour does not occur in number partitioning. In addition, they demonstrate that phase transition behaviour is useful for more than just simple benchmarking. It can, for instance, be used to analyse heuristics, and to compare the quality of heuristic solutions.

Key words: heuristics, number partitioning, phase transitions

1. INTRODUCTION

Where are the hard computational problems? Many instances of NP-complete problems are surprising easy to solve. One place to find hard instances is at a phase transition (Cheeseman, Kanefsky, & Taylor, 1991; Mitchell, Selman, & Levesque, 1992). Problems which are very under-constrained are soluble and it is usually easy to guess one of the many solutions. Problems which are very over-constrained are insoluble. In the phase transition in between, problems are "critically constrained" and it is typically very hard to determine if they are soluble or insoluble (Cheeseman et al., 1991). Problems from the phase transition are now routinely used to benchmark algorithms. In this paper, we show how phase transition behaviour can also be used in the study of heuristics. We define a parameter that measures the "constrainedness" of an ensemble of number partitioning problems and identify a phase transition at a critical value of constrainedness. We then show that constrainedness can be used to analyse and compare algorithms and heuristics for number partitioning in a precise and quantitative manner. We conjecture that a similar approach will prove useful in a wide variety of NP-complete problems.

The paper is structured as follows. In Section 2 we define number partitioning and outline its practical and theoretical importance. We describe some well known heuristics and algorithms for number partitioning in Sections 3 and 4. In Section 5, we argue that the performance of algorithms and heuristics on an ensemble of combinatorial problems depends on their "constrainedness". We then develop a simple "annealed" theory (Section 6) for computing the constrainedness of an ensemble of number partitioning problems. In Section 7 we show that a phase transition occurs around a critical value of constrainedness. Associated with this phase transition, there is a complexity peak in the cost of finding the optimal partition (Section 8). In Section 9 we demonstrate that constrainedness also predicts the location of a phase transition for finding less than perfect partitions. We then show that constrainedness can be used to model the size of the optimal partition difference (10), to compare the quality of heuristic solutions (Section 11), and to analyse heuristics themselves (Sections 12 and 13). Finally, we mention some related work (Section 14). This paper greatly extends results that first appeared in (Gent & Walsh, 1996a) and in one section of (Gent, MacIntyre, Prosser, & Walsh, 1996). In addition, it includes many new results (for example, much of Sections 8, 10, 11 and 12 and all of Section 13) that have not appeared before in any form.

2. NUMBER PARTITIONING

Let us partition a bag of n positive integers into two disjoint bags. The partition difference, Δ is the absolute difference between the sums of the two bags. The number partition decision problem is to determine if there is a partition such that $\Delta \leq d$ for some given d. The number partition optimization problem is to determine the minimum partition difference, Δ_{opt}. If $\Delta \leq 1$ then the partition is *perfect* otherwise we call it *imperfect*. As in (Korf, 1995), we consider n numbers drawn uniformly and at random from $(0, l]$. We have, however, seen very similar results with a Poisson distribution with parameter l.

Number partitioning is of considerable importance, both practically and theoretically. Many problems in AI (like scheduling and processor allocation, and the minimization of VLSI circuit size and delay) involve partitioning bags of numbers. Number partitioning is also one of Garey and Johnson's six basic NP-complete problems (Garey & Johnson, 1979). Phase transition behaviour has been observed in three out of these six problems: 3-SAT (Mitchell et al., 1992), CLIQUE via the dual independent set problem (Gent & Walsh, 1994), and HAMILTONIAN CIRCUIT (Cheeseman et al., 1991). Here, we show it occurring in a fourth problem, PARTITION. We predict that the two remaining problems, 3-DIMENSIONAL MATCHING and VERTEX COVER will also display similar phase transition behaviour.

Number partitioning is the only one of Garey and Johnson's six basic problem that deals with numbers. It is often therefore the natural choice for NP-completeness proofs of other problems involving numbers (*e.g.* bin packing, multiprocessor scheduling, open-shop scheduling, quadratic programming, and knapsack problems). Whilst number partitioning is a NP-complete problem, it is not NP-complete in the "strong" sense since the optimization problem (and hence the decision problem) can be solved in pseudo-polynomial time using dynamic programming (Garey & Johnson, 1979). Given n numbers to partition which sum to s, this algorithm runs in time and space bounded by a low polynomial in $n.s$. Note, however, that the size of the input need only be $O(n.\log(s))$. The NP-completeness of the PARTITION problem depends on the fact that we can supply extremely large integers to partition. For this reason, the PARTITION problem offers a strong test of the connection between phase transitions and NP-completeness.

Our results refute a conjecture of Y. Fu that "at least one NP complete problem, that of random number partitioning, can be solved exactly in statistical mechanics and no phase transition of any kind is found" (Fu, 1989). Fu later suggests that "Unless there is a subtle loophole in our argument, then we must accept as a conclusion that no connection whatsoever exists between the spin glass transition and NP-completeness ... the connection between phase transition and computational complexity, if any, must be of a form very different from what has been imagined". By means of an annealed theory, we identify a simple phase transition for the number partition decision problem. Number partitioning therefore does not provide evidence that phase transitions are unconnected with NP-completeness. It remains an open question if there is any NP-complete problem which lacks a phase transition. We should note, however, that phase transition behavior has been proven to occur in polynomial problems. For instance, 2-satisfiability displays a phase transition in solubility as we vary the ratio of clauses to variables (Chvatal & Reed, 1992).

3. HEURISTICS FOR NUMBER PARTITIONING

A variety of heuristics have been proposed for number partitioning. The greedy heuristic, for instance, simply places the largest remaining number into the bag with the smaller sum (Korf, 1995). Consider partitioning the bag $\{25, 17, 10, 8, 7, 4\}$ using the greedy heuristic:

Numbers remaining	Partial partition	Δ
$\{25, 17, 10, 8, 7, 4\}$	–	–
$\{17, 10, 8, 7, 4\}$	$\{25\}\{\}$	25
$\{10, 8, 7, 4\}$	$\{25\}\{17\}$	8
$\{8, 7, 4\}$	$\{25\}\{17, 10\}$	2
$\{7, 4\}$	$\{25, 8\}\{17, 10\}$	6
$\{4\}$	$\{25, 8\}\{17, 10, 7\}$	1
–	$\{25, 8, 4\}\{17, 10, 7\}$	3

The partition constructed by the greedy heuristic thus has a partition difference, Δ of 3.

The set differencing method of Karmarkar and Karp replaces two numbers by their difference (Karmarkar & Karp, 1982). This commits the two numbers to opposite bags without deciding into which bag each number goes. For example, consider again partitioning the bag $\{25, 17, 10, 8, 7, 4\}$ into two separate bags. If we put 25 in the first bag and 17 in the second bag, then this is equivalent to placing their difference, 8 in the first bag. On the other hand, if we put 17 in the first bag and 25 in the second, then this is equivalent to placing 8 in the second bag. Thus, if we decide that 25 and 17 are to go into opposite bags, we simply need to remove the two numbers and replace them by their difference, 8. We can then partition the bag $\{10, 8, 8, 7, 4\}$.

Karmarkar and Karp give various methods for selecting the numbers to difference. They first suggest that the numbers should be chosen so that they have a small difference. However, in algorithm A of (Karmarkar & Karp, 1982) they repeatedly difference the two largest numbers left in the bag. Following Korf we call this the KK heuristic (Korf, 1995). Consider partitioning the bag $\{25, 17, 10, 8, 7, 4\}$ using the KK heuristic:

Numbers remaining	Partial partitions	Δs
$\{25, 17, 10, 8, 7, 4\}$	–	–
$\{10, 8, 8, 7, 4\}$	$\{25\}\{17\}$	8
$\{8, 7, 4, 2\}$	$\{25\}\{17, 10\}$	2
$\{4, 2, 1\}$	$\{25\}\{17, 10\}$ and $\{8\}\{7\}$	2,1
$\{2, 1\}$	$\{25, 4\}\{17, 10\}$ and $\{8\}\{7\}$	2,1
–	$\{25, 7, 4\}\{17, 10, 8\}$	1

This is a perfect (and therefore optimal) partition. Indeed, when there are four or less numbers to partition, the KK heuristic is guaranteed to find the optimal partition difference (Korf, 1995). Korf states this result without proof, but we take the opportunity to present the proof since it identifies how larger problems can defeat the KK heuristic.

Theorem 1. Given a bag of n numbers with $n \leq 4$, the KK heuristic returns the optimal partition difference.

Proof. If $n = 2$, the KK heuristic puts the two numbers in opposite partitions. This is trivially optimal.

If $n = 3$, consider partitioning $\{a, b, c\}$ with $a \geq b \geq c$. The KK heuristic differences the largest two numbers, giving the bag $\{a - b, c\}$. There are two cases to consider. In the first case, assume that there is an optimal partition with a and b in opposite partitions. The optimal partition difference is then the same as that of the bag $\{a - b, c\}$. By the $n = 2$ case, the KK heuristic will construct this difference. In the second case, assume that all optimal partitions have a and b in the same partition. The best such partition has $\{a, b\}$ in one partition and $\{c\}$ in the other. This gives a partition difference, Δ_{a+b} of $(a + b) - c$. Consider swapping b with c so that a and b are in opposite partitions. This gives a partition difference, Δ_{a-b} of $(a + c) - b$. Now $\Delta_{a+b} = (a + b) - c \geq (a + b) - c - 2(b - c) = (a + c) - b = \Delta_{a-b}$. Thus, there exist partition differences the same size or smaller with a and b in oppositve partitions. This contradicts the assumption that all optimal partitions have a and b in the same partition. Hence, there is an optimal partition with a and b in opposite partitions and the KK heuristic will find it.

If $n = 4$, consider partitioning $\{a, b, c, d\}$ with $a \geq b \geq c \geq d$. The KK heuristic differences the largest two numbers, giving the bag $\{a - b, c, d\}$. There are two cases to consider. In the first case, assume that there is an optimal partition with a and b in opposite partitions. The optimal partition difference is then the same as that of the bag $\{a - b, c, d\}$. By the $n = 3$ case, the KK heuristic will construct this difference. In the second case, assume that all optimal partitions have a and b in the same partition. The best such partition has $\{a, b\}$ in one partition and $\{c, d\}$ in the other. This gives a partition difference, Δ_{a+b} of $(a + b) - (c + d)$. Consider swapping b with c so that a and b are in opposite partitions. This gives a partition difference, Δ_{a-b} of $(a + c) - (b + d)$. Now $\Delta_{a+b} = (a+b)-(c+d) \geq (a+b)-(c+d)-2(b-c) = (a+c)-(b+d) = \Delta_{a-b}$. Thus, there exist partition differences the same size or smaller with a and b in oppositve partitions. This contradicts the assumption that all optimal partitions have a and b in the same partition. Hence, there is an optimal partition with a and b in opposite partitions and the KK heuristic will find it. ■

Note that the largest two numbers always end up in opposite partitions. With 5 or more numbers to partition, the KK heuristic may not return the optimal partition difference. One obvious method to defeat the KK heuristic is to construct a problem in which the two largest numbers have to go in the same partition. The example given in (Korf, 1995) has this property. The bag $\{8, 7, 6, 5, 4\}$ has a perfect partition with the two largest numbers, $\{8, 7\}$ in one partition and $\{6, 5, 4\}$ in the other. On this problem, the KK heuristic puts $\{8, 6\}$ in one partition and $\{7, 5, 4\}$ in the other. This gives a partition difference of 2.

Karmarkar and Karp give other methods for selecting the numbers to difference. In algorithm B of (Karmarkar & Karp, 1982), they pair the largest two numbers, then the next largest two and so on. They difference each pair, thereby halving the number of numbers in the bag. They then re-pair the numbers and repeat. We call this the KK2 heuristic. Consider again partitioning the bag $\{25, 17, 10, 8, 7, 4\}$ using the KK2 heuristic:

Numbers remaining	Partial partitions	Δs
$\{25, 17, 10, 8, 7, 4\}$	–	–
$\{8, 2, 3\}$	$\{25\}\{17\}$ and $\{10\}\{8\}$ and $\{7\}\{4\}$	8, 2, 3
$\{5, 2\}$	$\{25, 4\}\{17, 7\}$ and $\{10\}\{8\}$	5, 2
–	$\{25, 8, 4\}\{17, 10, 7\}$	3

The partition constructed by the KK2 heuristic thus has a partition difference, Δ of 3.

All three set differencing heuristics run in $O(n \log n)$ time. For a large class of input distributions on $[0, 1]$, Karmarkar and Karp prove that a simple variant of the KK2 heuristic constructs a partition difference of size $O(1/n^{c \log n})$ for $c > 0$ with probability approaching 1 as $n \mapsto \infty$. Under similar probabilistic assumptions, the greedy heuristic cannot be expected to give a partition difference of size less than $O(1/n)$ (Karmarkar & Karp, 1982).

4. ALGORITHMS FOR NUMBER PARTITIONING

As mentioned in Section 2, there is a simple algorithm for the number partition optimization problem based upon dynamic programming which runs in pseudo-polynomial time. This algorithm uses a bit array with $\lceil s/2 + 1 \rceil$ elements where s is the sum of the numbers being partitioned. The jth bit in this array is set if there is a subset that sums to j. All possible subset sums are enumerated using a simple n step loop. Initially just the 0th bit in the array is set. On the ith step of the loop, the jth bit is set if k is the ith number to be partitioned and the $(j - k)$th bit was previously set. The bit array thus represents the sums of all possible subsets of the first i numbers. If the hth bit is the highest bit set after the nth iteration then the optimal partition difference, Δ_{opt} is $s - 2h$. A perfect partition exists iff the top bit in the array is set during one of the iterations. Unfortunately, this procedure requires space of size $O(n\, l)$ where l is the size of the numbers being partitioned, and this is exponential in the size of the input which is just $O(n \log(l))$. One modification is to represent just the non-zero bits in the partial sums. On certain problems, this may allow dynamic programming to be competitive with the other algorithms presented here.

Ruml *et al.* have performed a comprehensive study of stochastic approximation algorithms for number partitioning (Ruml, Ngo, Marks, & Shieber, 1994). They used several different search procedures including simulated annealing, hill climbing and a genetic algorithm, with a variety of different encodings of the number partitioning problem. They found that the choice of encoding was more important than the choice of search engine. A naive encoding with a bit for each number representing the bag into which it is put gave poor performance. However, more complex encodings based upon relaxing the greedy and KK heuristics gave good performance. With a small amount of search, they were able to find solutions several orders of magnitude better than that returned without search by the KK heuristic.

Korf has proposed a simple backtracking algorithm for finding the optimal partition difference based upon the greedy heuristic (Korf, 1995). This algorithm uses the greedy heuristic to branch, placing the largest remaining number into the bag with the smaller sum. On backtracking, we place the largest remaining number into

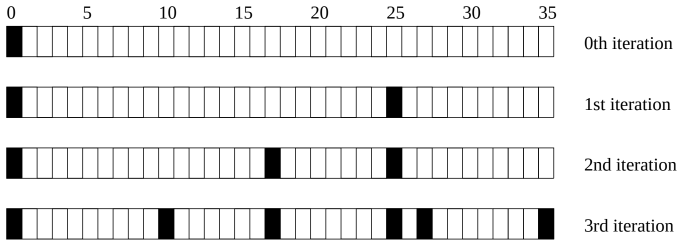

FIGURE 1. The use of dynamic programming to partition the bag $\{25, 17, 10, 8, 7, 4\}$. The sum of the numbers in the bag is 71 so we need a bit array with 36 elements. Initially just the 0th bit of the array is set. On the 1st iteration, we set the 25th bit as 25 is the 1st number to be partitioned and the 0th bit was set previously. On the 2nd iteration, we set the 17th bit as 17 is the 2nd number to be partitioned and the 0th bit were set previously. As the 25th bit was set in the first iteration, we add 17 to 25 but this takes us to the 42nd bit which is off the end of array. On the 3rd iteration, we set the 10th, 27th and 35th bits as 10 is the 3rd number to be partitioned and the 0th, 17th and 25th bits were set previously. Setting the 35th and top bit reflects the fact that the subset, $\{25, 10\}$ sums to 35. There is therefore a perfect partition, $\{25, 10\}$ and $\{17, 8, 7, 4\}$ with $\Delta_{opt} = 1$.

the bag with the larger sum. We terminate search if we discover a perfect partition or if the binary search tree is exhausted. The largest number is arbitrarily placed in the first bag. Similarly, if the bags have the same sum, we do not need to backtrack at this node. Finally, if the sum of the remaining numbers is ever less than or equal to the difference between the sum of the bags, the optimal solution to the current subproblem assigns all the remaining numbers to the smaller bag.

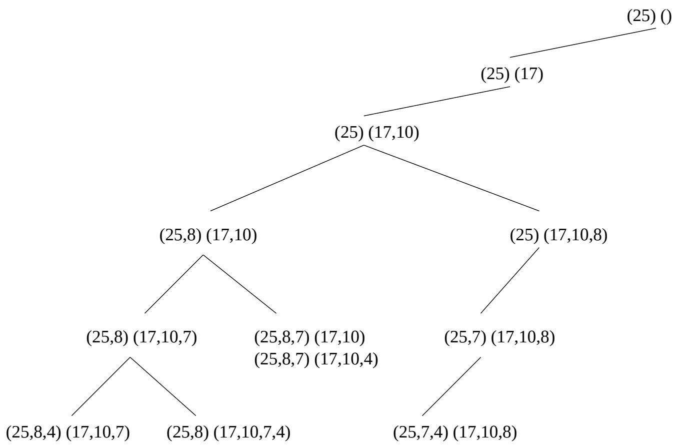

FIGURE 2. The search tree explored in a depth-first manner by Korf's greedy backtracking algorithm when finding the optimal partition difference for the bag $\{25, 17, 10, 8, 7, 4\}$.

Korf has also proposed the CKK algorithm for finding the optimal partition difference (Korf, 1995). This is similar to the greedy algorithm but uses the KK set differencing heuristic to branch. Initially we replace the largest two numbers by their difference. This commits to those subproblems in which the two largest numbers go into opposite bags. On backtracking, the only other choice is to replace the two largest numbers by their sum. This commits to those subproblems in which the two largest numbers go into the same bag. When there are four numbers left, we commit to the KK solution since this is optimal. We again terminate search if we discover a perfect partition or if the binary search tree is exhausted. Finally, if the largest remaining number is greater than or equal to the sum of the rest, the optimal solution to the current subproblem places the largest number in one bag and the rest of the numbers in the other bag.

Korf has shown that the CKK algorithm can give orders of magnitude better performance than the greedy backtracking algorithm. He claims that CKK outperforms all other algorithms in the literature, including the incomplete stochastic procedures described in (Ruml et al., 1994). The CKK algorithm can, for instance, partition arbitrarily large bags of integers with up to 12 decimal digits.

5. CONSTRAINEDNESS

The performance of an algorithm or a heuristic on an ensemble of combinatorial problems depends on the constrainedness of the problems. Problems that are "*critically constrained*" are on the knife-edge between solubility and insolubility. Such problems tend to be hard to solve as a lot of search is usually needed to determine if they have a solution or not. In collaboration with Patrick Prosser and Ewan MacIntyre, we have defined a *constrainedness* parameter, κ for an ensemble of combinatorial problems (Gent et al., 1996). If we have a state space of size 2^N in which $\langle Sol \rangle$ of the states are expected to be solutions then,

$$\kappa \quad =^{\text{def}} \quad 1 - \frac{\log_2(\langle Sol \rangle)}{N}. \tag{1}$$

The "1 -" simply rescales κ so that it lies in the interval $[0, \infty)$. If κ is small then problems in the ensemble are under-constrained and are likely to have a large number of solutions compared to the problem size. They therefore tend to be relatively easy to solve. If κ is large then problems in the ensemble are over-constrained and are likely to have very few or no solutions. The excess of constraints tends to make it relatively easy to show that such problems are insoluble. We predict that a phase transition will occur when $\kappa \approx 1$ and problems are on the "knife-edge" between solubility and insolubility. A lot of search is usually needed either to find a solution or to show that none exists. Note that if $\kappa = 1$ then $\langle Sol \rangle = 1$. Earlier studies have predicted a phase transition for constraint satisfaction problems at $\langle Sol \rangle \approx 1$ (Williams & Hogg, 1994; Smith & Dyer, 1996).

Although κ ignores specific features of the problem structure like clustering of solutions, it is surprisingly useful for a wide variety of problems. For example, for number partitioning the prediction of a phase transition at $\kappa \approx 1$ is accurate to within about 4% (Section 7). More refined predictions for the location of the phase boundary take account of the variance in the number of solutions at the phase boundary (Williams & Hogg, 1994; Smith & Dyer, 1996). There is a small but significant difference between the prediction of a phase transition at $\kappa \approx 1$ and the prediction of a phase transition at $\langle Sol \rangle \approx 1$. At the phase boundary, $\langle Sol \rangle$ can grow exponentially with problem size, whilst κ tends to vary very little. For example, at the phase transition for 3-satisfiability, $\langle Sol \rangle$ grows as approximately $2^{0.18N}$ where N is the number of variables whilst κ, which is proportional to the ratio of clauses to variables, remains relatively constant (Gent et al., 1996). What variation there is in κ at the phase transition can be modelled by the technique of finite size scaling described in Section 7. As a parameter, κ is therefore very useful for comparing problems of different sizes. Indeed, κ has been used to study phase transition behaviour in a wide variety of domains including constraint satisfaction, satisfiability, graph colouring, traveling salesperson and number partitioning problems (Gent et al., 1996).

6. ANNEALED THEORY

To compute the constrainedness of an ensemble of number partitioning problems drawn from a uniform distribution, we need to know the expected number of solutions. We approximate this by means of an annealed theory. In the next section, we show that this annealed theory gives good results in practice. Consider finding a perfect partition for a bag of n numbers drawn uniformly and at random from $(0, l]$. We restrict attention to bags with an even sum. A similar analysis can be given for bags with an odd sum. We want to partition the bag into two bags that add up to the same target sum (that is, half the sum of the bag being partitioned). We take the binary representation of the n numbers and average probabilities independently over the different digit positions. We call this an "annealed theory" by analogy with an annealed theory of materials which averages independently over sources of disorder. It provides a good approximation as n tends to infinity.

The least significant bits in each of the two bags must add up to a number with the same parity as the target. On average, we expect just 1/2 the possible partitions to give the same parity for the least significant bit as the target. We can apply the same argument to the bits and carries at each binary digit position, of which there are $\log_2(l)$. If we assume independence between bit positions, a partition is perfect in a fraction, $(1/2)^{\log_2(l)}$ (that is, $1/l$) of the 2^n possible partitions. The expected number of perfect partitions, $\langle Sol \rangle$ is therefore simply,

$$\langle Sol \rangle = \frac{2^n}{l}.$$

The choice of representing numbers in base 2 does not affect this result. In number base b, we expect each digit position to match modulo b with probability $1/b$, and there are $\log_b(l)$ digits base b. The expected number of perfect partitions therefore remains unchanged. Substituting n for N and the annealed estimate for $\langle Sol \rangle$ in Equation 1 and simplifying gives,

$$\kappa = \frac{\log_2(l)}{n}.$$

In the next section, we use this parameter to identify a phase transition. As in other problem classes, problems from the phase transition are useful for benchmarking algorithms and heuristics.

7. PHASE TRANSITION

In Figure 3 we plot the probability that a bag with an even sum has a perfect partition against κ for n from 6 to 30, and $\log_2(l)$ from 0 to $2n$. In this and all subsequent experiments, 1000 problems were generated at each value of l and n. Almost identical results are seen with bags with an odd sum, and with bags with both odd and even sums. As predicted in the last section, a phase transition occurs around $\kappa \approx 1$. In addition, the transition sharpens as n increases. This refutes Fu's conjecture that number partitioning lacks a phase transition (Fu, 1989).

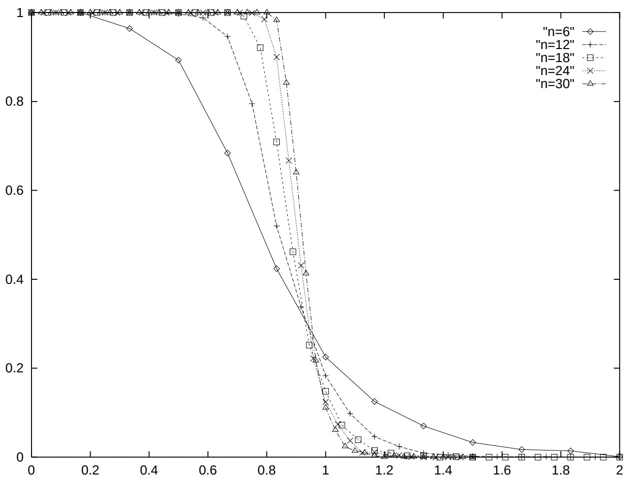

Figure 3. Probability of a perfect partition existing (y-axis) against κ (x-axis). Each problem has n numbers drawn uniformly and at random from $(0, l]$ with n from 6 to 30, and $\log_2(l)$ varied from 0 to $2n$.

We next applied finite size scaling methods (Barber, 1983) to determine how the probability scales with problem size. Around some critical point, we predict that problems of all sizes will be indistinguishable except for a change of scale given by a simple "power law". This suggests,

$$Prob(\text{perfect partition}) = f\left(\left(\frac{\kappa - \kappa_c}{\kappa_c}\right) \cdot n^{1/\nu}\right) \tag{2}$$

where f is some fixed function, κ_c is the critical point, and $n^{1/\nu}$ is the power law that provides the change of scale. The fraction, $(\kappa - \kappa_c)/\kappa_c$ plays the rôle of the reduced temperature, $(T - T_c)/T_c$ in physical systems. The size dependency is given by a simple power law, $n^{1/\nu}$. In physical systems, the exponent $1/\nu$ can often be calculated exactly. Equation 2 has a fixed point where κ equals κ_c and for all n, the probability is the constant value $f(0)$. To estimate κ_c, we take the fixed point to be the point with the minimum spread in probabilities. This gives $\kappa_c = 0.96 \pm 0.02$, where the errors indicate the range giving less than 9% spread. To compute ν, we assume that Equation 2 holds at the point of 50% probability, and calculate the median estimate for ν. This gives $\nu = 1 \pm 0.3$ where errors represent the upper and lower quartiles of estimates of ν. This error may appear large, but the quality of the fit is relatively insensitive to the exact value of ν. We define a rescaled parameter,

$$\gamma =^{\text{def}} \frac{\kappa - \kappa_c}{\kappa_c} \cdot n^{1/\nu}.$$

In Figure 4, we plot the probability of a perfect partition existing against γ with

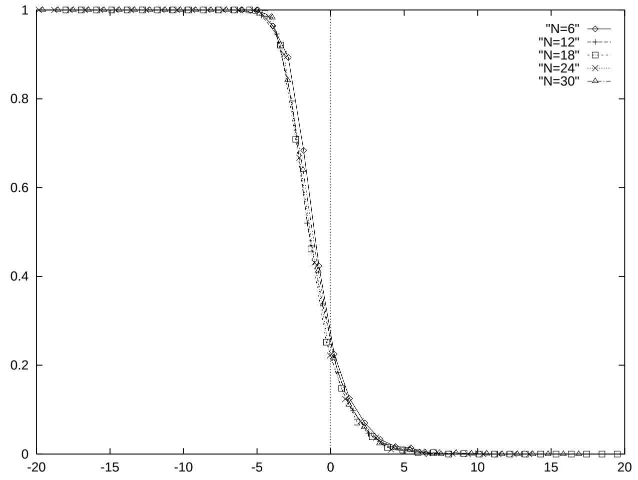

FIGURE 4. Probability of a perfect partition existing (y-axis) against γ (x-axis) with $\kappa_c = 0.96$ and $\nu = 1$. Each problem has n numbers drawn uniformly and at random from $(0, l]$ with n from 6 to 30, and $\log_2(l)$ varied from 0 to $2n$.

$\kappa_c = 0.96$ and $\nu = 1$. This graph suggest that finite size scaling provides both a simple and accurate model for the scaling of probability with problem size. A similar rescaling describes the finite size scaling of the phase transition in satisfiability (Kirkpatrick & Selman, 1994), constraint satisfaction (Gent, MacIntyre, Prosser, & Walsh, 1995) and traveling salesperson problems (Gent & Walsh, 1996b).

8. OPTIMIZATION COST

Like many other combinatorial problems, a peak in the cost to find the optimal solution is associated with the phase transition in solubility. The phase transition is thus a useful source of benchmark problems. In Figure 5, we plot the average number of nodes searched by the CKK algorithm to find the optimal partition difference against the rescaled parameter, γ with $\kappa_c = 0.96$ and $\nu = 1$, n fixed at values between 6 and 30 and $\log_2(l)$ varying from 0 to $2n$. We have observed a similar result for Korf's greedy backtracking algorithm. As in satisfiability (Selman & Kirkpatrick, 1996), constraint satisfaction (Gent et al., 1995), and the traveling salesperson problem (Gent & Walsh, 1996b) finite size rescaling offers a clear and consistent view of how search cost varies through the phase transition.

In the soluble phase problems are, on average, easy. Problem hardness increases as we approach the phase boundary. The cost of finding the optimal partition difference remains uniformly hard in the insoluble phase away from the phase boundary. Experiments out to larger l do not show problems becoming significantly easier (or harder) well away from the phase transition. This reflects the fact that, for fixed n, a lot of search is needed by the Ckk algorithm to prove that the optimal solution is optimal even for very over-constrained problems that have a large optimal partition difference.

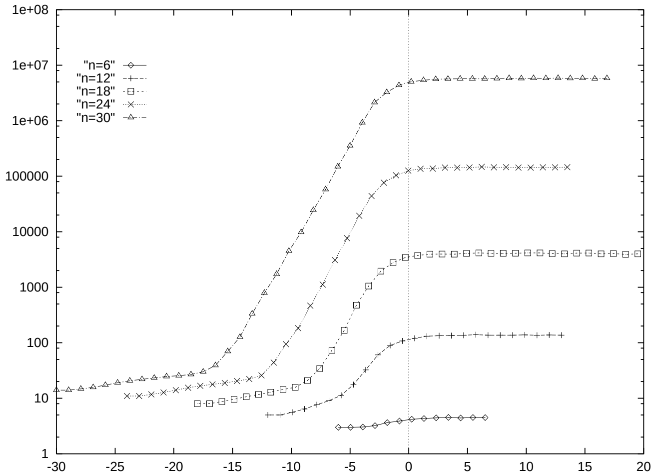

Figure 5. Average nodes searched by Ckk to find the optimal partition difference (y-axis) against γ (x-axis) for fixed n with $\kappa_c = 0.96$ and $\nu = 1$. Each problem has n numbers drawn uniformly and at random from $(0, l]$ with n from 6 to 30, and $\log_2(l)$ varied from 0 to $2n$.

Another view of the phase transition comes from fixing l and varying n. We now observe an easy-hard-easy pattern (see, for example, Figure 5 in (Korf, 1995)). In the insoluble region, problems become easier away from the phase boundary since n, and thus the total number of partitions, is decreasing. If we fix l and vary n or fix n and vary l, the size of the input problem (the number of bits needed to specify a problem) varies. A third view of the phase transition is thus to vary the constrainedness, κ but keep the input problem size, $n \log_2(l)$ constant. In Figure 6, we plot the average number of nodes searched by the Ckk algorithm to find the optimal partition against the rescaled parameter, γ with $\kappa_c = 0.96$ and $\nu = 1$, and $n \log_2(l)$ fixed at 6^2, 12^2, 18^2 and 24^2. In each case, we varied n in steps of 1 from 36 downwards. As expected, the phase transition in solubility occurs around $\gamma = 0$. In addition, we now see an easy-hard-easy pattern. Problems in the insoluble region again become easier as we move away from the phase boundary since n, the size of the bags being partitioned decreases.

In Figure 7 we plot the maximum value of the mean search cost when we fix n and vary l for both the Ckk and the greedy backtracking algorithms (Korf, 1995). From the gradient of these graphs, we estimate that the worst average search costs grow as approximately $2^{0.85n}$ for the Ckk algorithm and approximately $2^{0.90n}$ for the

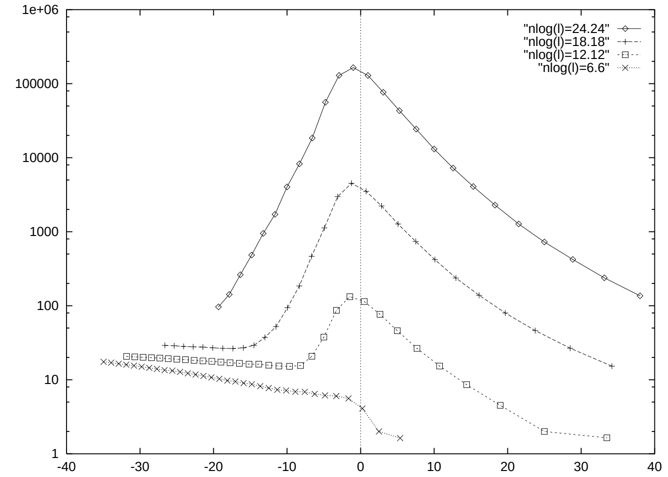

FIGURE 6. Average nodes searched by CKK to find the optimal partition (y-axis) against γ (x-axis) with $\kappa_c = 0.96$ and $\nu = 1$, and the problem input size, $n \log_2(l)$ fixed at 6^2, 12^2, 18^2 and 24^2. Each problem has n numbers drawn uniformly and at random from $(0, l]$ with n varied from 1 to 36.

greedy backtracking algorithm. Note that simply computing all possible partitions would give a maximum search costs that grows as 2^n. This confirms quantitatively Korf's claim that "CKK is asymptotically more efficient than the standard [greedy backtracking] algorithm" (Korf, 1995).

9. IMPERFECT PARTITIONS

Phase transition behaviour is not restricted to the decision problem of finding a perfect partition. Constrainedness can also be used to identify a phase transition for the decision problem of finding an imperfect partition. We repeat our construction of an annealed theory to identify a constrainedness parameter for partitioning into imperfect partitions. Consider finding a partition difference of size d or less for a bag of n numbers drawn uniformly and at random from $(0, l]$. For simplicity, we assume that d is a power of 2. Since we want to find a partition difference of size d or less, we can simply ignore the bottom $\log_2(d)$ bits in each bag. We do, however, insist that the top $\log_2(l) - \log_2(d)$ bits in each bag add up to a particular parity. Using a similar annealed argument to Section 6, we get $\langle Sol \rangle = 2^n/(l-d)$. Substituting this value into Equation (1) gives,

$$\kappa = \frac{\log_2(l/d)}{n}.$$

Note that, as required, when $d = 1$ this reduces to $\log_2(l)/n$, the constrainedness of perfect number partitioning problems.

A phase transition again occurs around the value $\kappa \approx 1$. This phase transition rescales identically to that for perfect partitioning. In Figure 8, we plot the probab-

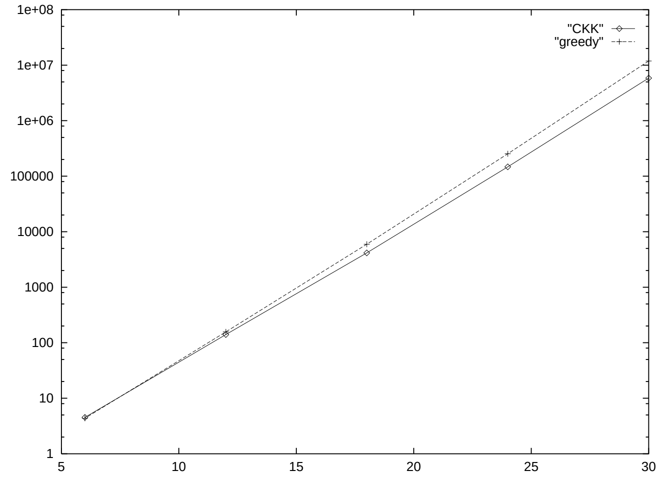

FIGURE 7. Maximum value over all l for the mean number of nodes searched to find the optimal partition difference (y-axis) against n (x-axis). Each problem has n numbers drawn uniformly and at random from $(0, l]$ with n from 6 to 30, and $\log_2(l)$ varied from 0 to $2n$.

ility that a bag has an imperfect partition against κ for $n = 24$, $\log_2(d)$ from 1 to 5 and 10, and $\log_2(l)$ from 0 to $2n$.

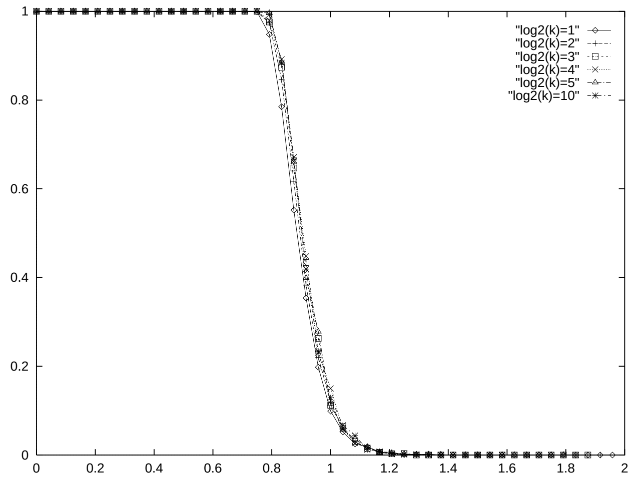

FIGURE 8. Probability of an imperfect partition of size d or less (y-axis) against κ (x-axis). Each problem has 24 numbers drawn uniformly and at random from $(0, l]$, with $\log_2(d)$ varied from 1 to 5 and 10, and $\log_2(l)$ varied from 0 to $2n$.

Search cost again peaks at the phase transition. In Figure 9, we plot the average number of nodes searched by the CKK algorithm against κ, again for $n = 24$, $\log_2(d)$ from 1 to 5 and 10, and $\log_2(l)$ from 0 to $2n$. We modify the CKK algorithm so that it runs as a decision procedure, terminating search immediately a partition less than or equal to d is found. Note that all problems in this graph are the same size, n so that we do not need to rescale.

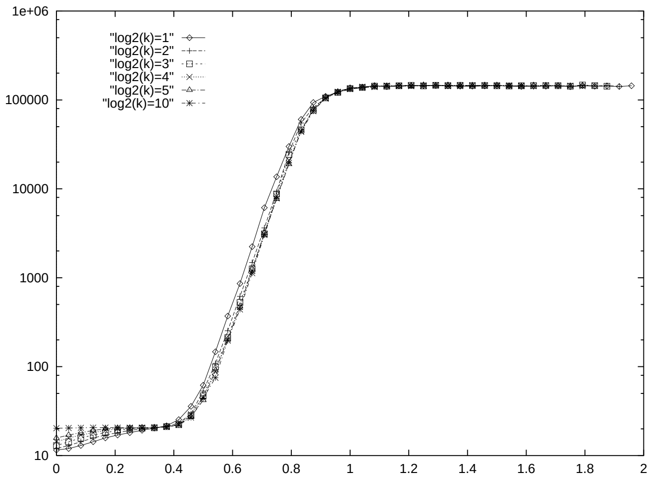

FIGURE 9. Average number of nodes searched by CKK finding an imperfect partition of size d or less (y-axis) against κ (x-axis). Each problem has 24 numbers drawn uniformly and at random from $(0, l]$, with $\log_2(d)$ varied from 1 to 5 and 10, and $\log_2(l)$ varied from 0 to $2n$.

10. OPTIMAL PARTITIONS

Finite size scaling is also a good method for modelling the size of the optimal partition difference. In Figure 10, we plot the mean optimal partition difference against the rescaled parameter, γ. For $\gamma \ll 0$, all problems have a perfect partition; as half the problems have an even sum and half have an odd sum, $\langle \Delta_{optimal} \rangle = 1/2$. (Karmarkar, Karp, Lueker, & Odlyzko, 1986) gives a simple heuristic argument for partitioning real numbers in $[0, 1]$ which we can adapt to estimate the optimal partition difference for integers drawn uniformly from $(0, l]$. By considering a random walk with steps of size l, we expect the sum of each partition to lie within an interval of size $O(l\sqrt{n})$. We divide this into 2^n sub-intervals, each of size $O(l\sqrt{n}/2^n)$. By a pigeonhole argument, we can expect to find two bags in the same sub-interval. That is, within $O(l\sqrt{n}/2^n)$ of each other. Hence, the mean optimal partition difference is of size $O(l\sqrt{n}/2^n)$. Assume that $\langle \Delta_{opt} \rangle \propto l\sqrt{n}/2^n$. In addition, assume that $\log_2(l) \gg \frac{1}{2}\log_2(n)$, $\kappa_c \approx 1$ and $\nu = 1$. Then $\log_2(\langle \Delta_{opt} \rangle) \propto \log_2(l\sqrt{n}/2^n) = \log_2(l) + \frac{1}{2}\log_2(n) - n \approx \log_2(l) - n = (\log_2(l)/n - 1)\, n \approx (\frac{\kappa - \kappa_c}{\kappa_c})\, n = \gamma$. Hence $\log_2(\langle \Delta_{opt} \rangle)$ is approximately proportional to the rescaled parameter, γ. This agrees with the data in Figure 10 since $\log_2(\langle \Delta_{opt} \rangle)$ plotted against γ gives a straight line with a gradient of $+1$.

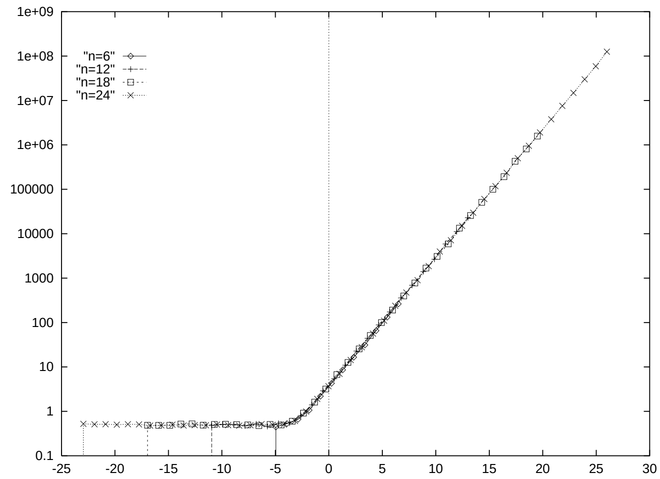

Figure 10. Mean optimal partition difference, $\langle\Delta_{opt}\rangle$ (y-axis) against γ (x-axis) with $\kappa_c = 0.96$ and $\nu = 1$. Each problem has n numbers drawn uniformly and at random from $(0, l]$, with n varied from 6 to 24 and $\log_2(l)$ varied from 0 to $2n$.

11. HEURISTIC PARTITIONS

The constrainedness parameter, κ also provides a quantitative method for comparing heuristics. Indeed, we can even use the finite size scaling of κ to compare heuristics. However, we need to substitute different values of κ_c into the definition of γ for different heuristics. These new values for κ_c reflect the quality of the heuristics as they are the minimum value of constrainedness for which the heuristic fails to find an optimal solution. In Figure 11, we plot the mean size of the KK partition difference (that is, the possibly sub-optimal partition difference found by the KK heuristic) against γ for $n = 6$ to 24 and $\log_2(l)$ from 0 to $2n$. To estimate κ_c, we again found the point with the minimum spread in probabilities. This gives $\kappa_c = 0.40$ and not 0.96 as previously. However, we continue to obtain a good fit with the same power-law coefficient, $\nu = 1$.

Recall that κ_c is the fixed point for the rescaling. It gives the value of the constrainedness parameter at which we move between different phases. For $\kappa < 0.40$, the KK heuristic almost always returns the optimal partition difference. For $\kappa > 0.40$, the quality of the solution returned decreases as n increases. In the region $0.40 < \kappa < 1$, the KK heuristic performs poorly as n increases. This is despite the fact that these problems usually have perfect partitions.

Performance guarantees for optimization procedures are often given as a ratio of the optimal value. Since the optimal partition difference can be zero, such a ratio can be undefined for perfect partitions. As in (Williams & Hogg, 1994), we make a "mean-field" approximation that, $\langle\frac{\Delta_{KK}}{\Delta_{opt}}\rangle \approx \frac{\langle\Delta_{KK}\rangle}{\langle\Delta_{opt}\rangle}$. The mean optimal partition difference, $\langle\Delta_{opt}\rangle$ is at least $1/2$ so the performance ratio is always defined. Another approach used in the literature is to add a small constant to the partition difference to prevent division by zero. For instance, we could measure the ratio, $(1 + \Delta_{KK})/(1 + \Delta_{opt})$.

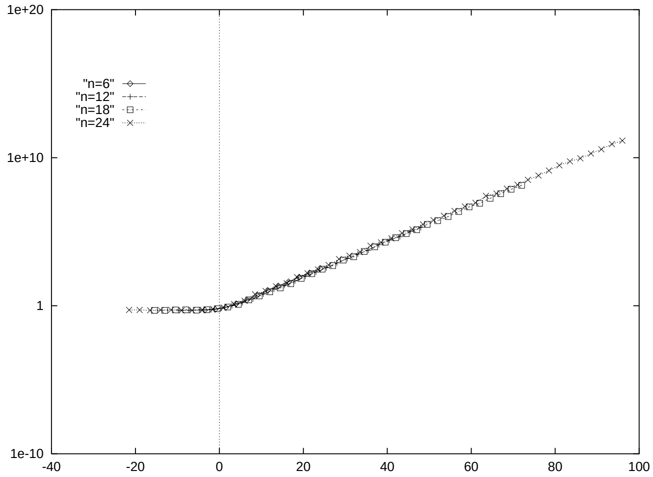

FIGURE 11. Mean size of the KK partition difference, $\langle\Delta_{KK}\rangle$ (y-axis) against γ (x-axis) with $\kappa_c = 0.40$ and $\nu = 1$. Each problem has n numbers drawn uniformly and at random from $(0, l]$, with n varied from 6 to 24 and $\log_2(l)$ varied from 0 to $2n$.

This gives similar results to the mean-field approximation.

In Figure 12, we plot the average performance ratio for the KK heuristic against γ again with $\kappa_c = 0.40$ and $\nu = 1$, $n = 6$ to 24 and $\log_2(l)$ from 0 to $2n$. The maximum performance ratio grows approximately as a simple exponential in n. We see very similar behaviour with the KK2 and greedy heuristics. However, for both heuristics, we need to rescale around $\kappa_c \approx 0.15$. Finite size scaling therefore provides us with a very simple and quantitative method for comparing heuristics. In the region $\kappa < 0.15$ all the heuristics return the optimal partition difference. In the region $0.15 < \kappa < 0.40$, the greedy and KK2 heuristics (but not the KK heuristic) perform poorly as n increases. This is despite the fact that these problems have perfect partitions that are usually found by the KK heuristic. And in the region $0.40 < \kappa < 1$, all the heuristics perform poorly as n increases, again despite the fact that these problems have perfect partitions that CKK can usually find with little search.

In addition to determining the range of constrainedness over which a heuristic returns the optimal partition difference, we can also plot how much worse it does than the optimal. That is, we can plot the constrainedness of the problems we wanted to solve (namely, finding a perfect partition with $\Delta \leq 1$) against the constrainedness of the problems the heuristic typically managed to solve (namely, finding an imperfect partition with $\Delta \leq d$ where d is the mean heuristic partition difference). In Figure 13, we plot the constrainedness of finding a perfect partition against the constrainedness of the heuristic partition found for three different heuristics: greedy, KK and KK2. In each experiment, 1000 bags of 24 numbers were generated at random from $(0, l]$ for each point from $\log_2(l) = 1$ to 48 in steps of 1. For comparison, we also plot the constrainedness of the best decision problem that is soluble, i.e. the value of κ when d is set to Δ_{opt}, the optimal partition difference.

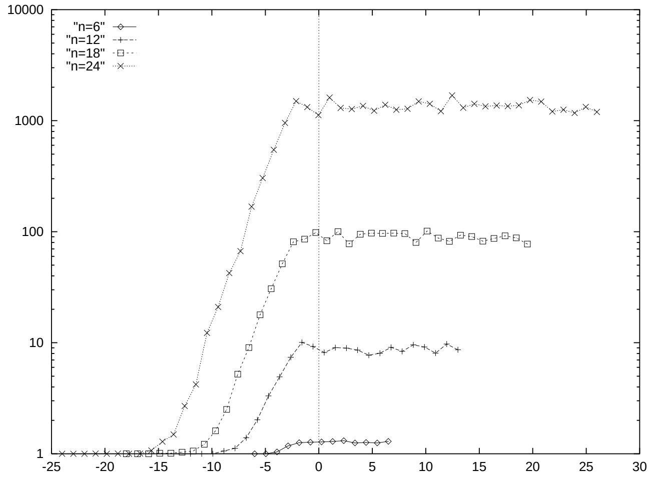

FIGURE 12. Average performance ratio for the KK heuristic, Δ_{KK}/Δ_{opt} (y-axis) against γ (x-axis) with $\kappa_c = 0.40$ and $\nu = 1$. Each problem has n numbers drawn uniformly and at random from $(0, l]$ with n varied from 6 to 24 and $\log_2(l)$ varied from 0 to $2n$.

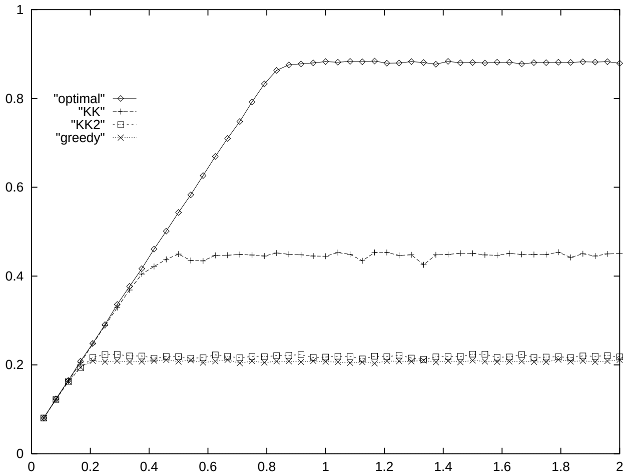

FIGURE 13. Average κ of the number partitioning problem achieved by a heuristic (y-axis) against κ of the desired perfect number partitioning problem (x-axis) for three heuristics and for the optimal partition difference. Each problem has 24 numbers drawn uniformly and at random from $(0, l]$, with $\log_2(l)$ varied from 1 to 48.

As before, for $\kappa < 0.40$, the KK heuristic almost always returns the optimal and perfect partition. For $\kappa > 0.40$, the quality of the solution returned decreases as $\log_2(l)$ increases, until $\kappa \approx 1$ where KK is consistently able to achieve a value of κ

about 0.5 less than the optimal. By comparison, the KK2 heuristic only performs well for $\kappa < 0.2$ and in the worst case does about 0.7 less than the optimal and about 0.2 worse than KK. The greedy heuristic performs just a little worse than KK2. The KK heuristic thus returns partition differences that are, in the worst case, a factor $2^{0.5n}$ larger than the optimum, whilst the KK2 and greedy heuristics return partition differences approximately $2^{0.7n}$ larger than the optimal. This graph clearly shows that KK is the best heuristic throughout the phase space, followed by the KK2 heuristic with the greedy heuristic very slightly behind.

12. CONSTRAINEDNESS AS A HEURISTIC

In the last section, we showed that κ provides a good method for comparing heuristics. Constrainedness can also be used to analyse heuristics. Gent, MacIntyre, Prosser and Walsh have recently suggested that many heuristics branch into the subproblem that minimizes the constrainedness, κ (Gent et al., 1996). The intuition is that we try to branch on the most constrained variable giving the most under-constrained and soluble subproblem. This viewpoint is useful for understanding the design of heuristics in constraint satisfaction (Gent, MacIntyre, Prosser, B.M.Smith, & Walsh, 1996) and, as we show here, in number partitioning.

Consider, for example, the KK heuristic. Recall that this heuristic takes a bag S of n numbers to partition and reduces it to a new bag R by removing the largest two numbers x and y, and replacing them by $x - y$ (we assume without loss of generality that $x \geq y$). This commits us to those solutions in which x and y are in opposite bags. Let $s = \sum_{i \in S} i$ and $r = \sum_{i \in R} i$. To compute κ for the subproblems generated by the KK heuristic, we again use an annealed theory. We assume that the size of the numbers, l is approximated by twice the mean. Experiments with non-uniform distributions of numbers suggest that κ for a wide class of distributions can be approximated by $\log_2(l)/n$, where l is the size of the numbers being partitioned. As $r = s - x - y + (x - y) = s - 2y$, the constrainedness κ goes from $\frac{\log_2(2s/n)}{n}$ to $\frac{\log_2(2(s-2y)/n-1)}{n-1}$. As we have no control over the denominator, κ is minimized by maximizing y. Given that $x \geq y$, the maximum y is the second largest element of S. And thus the KK heuristic minimizes κ by picking the two largest elements of S for x and y.

Karmarkar and Karp suggest that the motivation behind set differencing is to pick x and y so that $x - y$ is small (Karmarkar & Karp, 1982). We therefore ran experiments with a third set differencing heuristic, KK3 which choses x and y so that $x - y$ is minimal. This gave much poorer performance than the KK, KK2 and greedy heuristics. Minimizing constrainedness provides an explanation for the superiority of the KK heuristic over the KK2 heuristic, and of the KK2 heuristic over the KK3 heuristic. Unlike the KK heuristic, by working in phases the KK2 heuristic may not difference the two largest numbers. Consider, for instance, when the difference between the two largest numbers is larger than the smallest number (this occurs in the example in Section 3). The KK2 heuristic therefore reduces κ less than the KK heuristic. Whilst the KK2 often differences large numbers, the KK3 heuristic will frequently difference small numbers as such numbers are usually closer together. On average, the KK3 heuristic tends to reduce κ the least.

The greedy heuristic can also be seen as making decisions that minimizes κ. Although both the greedy and KK heuristics minimize κ, the decisions made by the greedy heuristic – assigning numbers to particular partitions – tend to be more constraining than the decisions made by the KK heuristic. We suggest that this helps to explain the superior performance of the KK heuristic over the greedy heristic. The analysis of the effect of the greedy heuristic on κ is a little more complex that that of set differencing heuristics like KK since the greedy heuristic builds a partial partition. We can factor this into our analysis by observing that if we have partitioned numbers into two bags, R and T with sums r and t and have a bag S of numbers remaining to be partitioned then this is equivalent to partitioning the bag $S \cup \{r-t\}$ (without loss of generality we assume that $r \geq t$). The greedy heuristic picks an element x of S and adds it to R or T. We minimize κ by maximizing the reduction in σ, the sum of $S \cup \{r-t\}$. If we add x to the bigger bag, R, then σ is unchanged. If, however, we add x to the smaller bag, T then σ decreases. This is what we therefore do. There are two cases to consider. These depend on whether $x > r-t$ or $x \leq r-t$. If $x > r-t$ then adding x to T makes it the partition with the larger sum, and σ reduces by $x+r-t-((x+t)-r) = 2(r-t)$. If, however, $x \leq r-t$, then adding x to T leaves it the bag with the smaller sum, and σ reduces by $x+r-t-(r-(x+t)) = 2x \leq 2(r-t)$. The first difference is always the larger and is therefore preferred. This suggests the heuristic of picking an x in S such that $x \geq r-t$, or failing that picking the largest x in S. The greedy heuristic does just this by picking the largest x in S and putting it in the smaller bag.

To test this hypothesis, we implemented a cautious heuristic which puts the smallest number left into the smaller bag. This compares to the greedy heuristic which puts the largest number left into the smaller bag. The intuition behind the cautious heuristic is to leave as much space as possible in two bags. We generated 1000 bags of 24 numbers at random from $(0, l]$ for each point from $\log_2(l) = 1$ to 48 in steps of 1. The cautious heuristic performed poorly on these problems. In almost every case, it failed to find the optimal partition difference. In the worst case, it returned partition differences that are a factor $2^{0.85n}$ larger than the optimum, and $2^{0.15n}$ larger than those returned by the greedy heuristic.

13. COMPARING HEURISTICS

We can compare analytically how good the greedy and KK heuristics are at reducing constrainedness. To do this, we contruct an "adaptive" heuristic that chooses between greedy and KK decompositions according to which reduces κ most. As shown by the following argument, such a heuristic is no better at reducing constrainedness than the KK heuristic. Let the target difference be the difference between the sums of the two bags. Simple analysis show that if the target difference is greater than the second largest number left to partition, then κ is decreased most if a greedy decomposition is performed, placing the largest number in the bag with the smaller sum. If not, κ is decreased most by a KK decomposition. If the target difference starts out zero, the adaptive heuristic will always apply the KK decomposition rule since the target difference will remain zero and this is always less than the second largest number left. We can, however, start by committing one number arbitrarily to the first bag. Such a commitment leaves κ unaffected but creates a target difference. It also reduces search since we do not consider isomorphic partitions in which the first number is placed in the second bag. We will maximize the decrease in κ at the second step if we begin by committing the largest number to the first bag. Although the adaptive heuristic will perform some greedy decompositions, it ultimately finds the same partition difference as the KK heuristic.

Theorem 2. The partition difference found by the adaptive heuristic given an initial target difference Δt and a bag S is identical to that found by the KK heuristic on the bag $S \cup \{\Delta t\}$.

Proof. By induction on the size of S. In the base case, S is empty and both heuristics return empty partitions. In the step case, let S be a bag containing $n+1$ numbers. We do a case split on the first decomposition step performed by the adaptive heuristic.

If the first step is a KK decomposition then the adaptive heuristic reduces the problem to one with $S - \{x, y\} \cup \{|x - y|\}$ where x, y are the largest elements of S. We can now appeal to the induction hypothesis as the new bag contains just n numbers. The adaptive heuristic constructs the same partition difference as KK on this smaller bag. And thus, as the first step is a KK decomposition for both the adaptive and KK heuristics, the same partition differences are constructed from S by both the adaptive and KK heuristics.

If the first step is a greedy decomposition then the adaptive heuristic reduces the problem to partitioning $S - \{x\}$ where x is the largest element of S with a new target difference of $|x - \Delta t|$. Since the adaptive heuristic performed a greedy decomposition, the second largest element of S is smaller than Δt. Given the initial bag, $S \cup \{\Delta t\}$, the KK heuristic takes the two largest elements, which must be x and Δt and replaces them by their difference $|x - \Delta t|$. But this corresponds to the same decomposition as the adaptive heuristic. We can now appeal to the induction hypothesis as the new bag, $S - \{x\}$ contains n numbers. The adaptive heuristic constructs the same partition difference as the KK heuristic on this smaller bag. Hence the same partitions are constructed from S by both the adaptive and KK heuristics. ∎

This result can be summarized by the slogan,

$$\text{KK} + \text{GREEDY} \quad = \quad \text{KK}.$$

Together with our earlier results that demonstrated the inferiority of the greedy heuristic compared to the KK heuristic, this shows that, for numbers drawn from a uniform distribution, the KK heuristic dominates the greedy heuristic. Of course, it is possible to construct individual problems or ensembles of problems on which the greedy heuristic outperforms the KK heuristic.

A slightly more complex argument shows that using the adaptive heuristic within a backtracking procedure gives an algorithm which finds partition differences in the same order as the CKK algorithm. The adaptive algorithm commits the largest number, x to the first bag. This creates a target difference, Δt of x. We then chose between greedy and KK decompositions according to which reduces κ most. A greedy split is applied if the target difference is greater than the second largest number left to partition, otherwise a KK split is performed. On backtracking, we perform the dual split. For example, a greedy split places the largest number in the bag with smallest sum. On backtracking, we place the largest number in the bag with largest sum. Search is pruned (as in CKK algorithm) if the largest number left including the target difference is greater than the sum of the remaining numbers.

Let S be the numbers remaining to be partitioned at a given node in the search tree of the adaptive algorithm, and Δt be the target difference at this point. If it is possible to follow the same path in the search tree of the CKK algorithm and R

is the set of numbers remaining to be partitioned by CKK at this point then we say that the nodes in the two trees are *equivalent* iff $R = S \cup \{\Delta t\}$.

Theorem 3. The nodes of the search tree of the adaptive algorithm are equivalent to those of the CKK algorithm.

Proof. Let S be the bag being partitioned. The proof uses induction on S. The base case is trivial. In the step case, let S be a bag containing $n+1$ numbers. We now do a case split on whether the first step along the path to a given node is to the left (i.e. with the heuristic) or right (i.e. against the heuristic).

If the first step is agreement with the heuristic, then we do a case split on whether the adaptive algorithm applies a greedy or a KK reduction. If it is a KK reduction then we reduce the problem to one of size n and appeal to the induction hypothesis. CKK trivially performs the same reduction. If the first step is a greedy reduction then the problem reduces to $S - \{x\}$ where x is the largest element of S, giving a new target difference of $|\Delta t - x|$. Since the greedy heuristic is applied, the second largest element of S is smaller than Δt. Given the initial bag, $S \cup \{\Delta t\}$, the CKK algorithm takes the two largest elements (which must be x and Δt) and replaces them by their difference. But this is the same reduction as the adaptive algorithm. We then appeal to the induction hypothesis on the reduced problem contains just n numbers.

If the first step is against the heuristic, then we do a case split on whether the adaptive algorithm applies the dual of a greedy or a KK reduction. If it is the dual of a KK reduction then we reduce the problem to one containing n numbers and appeal to the induction hypothesis. CKK trivially performs the same reduction. If the first step is the dual of a greedy reduction then the problem reduces to $S - \{x\}$ where x is the largest element of S, giving a new target difference of $x + \Delta t$. Since the dual of the greedy heuristic is applied, the second largest element of S is smaller than Δt. Given the initial bag, $S \cup \{\Delta t\}$, the CKK algorithm takes the two largest elements (which must be x and Δt) and replaces them by their sum. But this is the same reduction as the adaptive algorithm. We then appeal to the induction hypothesis on the reduced problem contains just n numbers.

Note that the pruning rules for the CKK and adaptive algorithms are equivalent and act on equivalent nodes. The search trees are therefore an identical shape. Hence, the two algorithms enumerate partitions in the same order. ■

This result can be summarized by the slogan,

$$\text{CKK} + \text{GREEDY} \quad = \quad \text{CKK}.$$

Together with our earlier results that demonstrated the inferiority of the greedy backtracking algorithm compared to the CKK algorithm, this shows that, for numbers drawn from a uniform distribution, the CKK algorithm dominates the greedy backtracking algorithm.

14. RELATED WORK

Unlike the simple annealed theory presented here, the exact analysis of number partitioning problems has proved difficult. Karmarkar, Karp, Lueker and Odlyzko have determined bounds on the probability distribution for the size of the optimal partition difference for a bag of real numbers drawn from the interval $[0, 1]$ (Karmarkar et al., 1986). Using some complicated analysis based on second moments, they proved that the median optimal partition difference is of size $\Theta(\sqrt{n}/2^n)$. When partitions are restricted to those of equal cardinality, the median optimal partition difference is of size $\Theta(n/2^n)$.

Karmarkar *et al.* admit that they were unable to derive the mean optimal partition difference. In addition, their results only apply to probability distributions that have a bounded density (for example, real numbers drawn uniformly from $[0, 1]$). They fail to hold for probability distributions in which there are values with non-zero probabilities (for example, integers drawn uniformly and at random from $(0, l]$). Korf (personal communication) has predicted that a phase transition occurs when the median optimal partition difference is 1, and observed that this coincides with a peak in search cost. Using the asymptotic value for the median optimal partition difference, Korf suggested that the phase transition occurs when $\sqrt{n}l/2^n = 1$. This agrees asymptotically with $\kappa = 1$.

We have extended the annealed theory presented in Section 6 to multi-way partitioning (Gent & Walsh, 1996a). For m-way partitioning problem, we show that the constrainedness κ is $(m-1)\log_m(l)/n$. As required, this reduces to $\log_2(l)/n$ for 2-way number partitioning. We have observed a phase transition for 3-way number partitioning around $\kappa \approx 1$. Search cost to find the optimal partition difference again peaks at the phase boundary.

Tad Hogg has used phase transition behaviour to inform the design of value ordering heuristics for graph colouring (Hogg, 1995). He applies the Brelaz heuristic to select the most constrained variable. An estimate of position with respect to the phase transition is then used to order values for this variable. He shows that this approach gives good performance on under-constrained and over-constrained problems but is less good at the phase transition.

Joseph Pemberton and Weixiong Zhang have exploited phase transition behaviour to solve combinatorial optimization problems approximately (Pemberton & Zhang, 1996). On a random tree model, their ϵ-transformation method runs in expected polynomial time, returning a solution with constant relative error. They also report good performance for approximating the asymmetric traveling salesperson problem and the maximum satisfiability problem.

15. CONCLUSIONS

We have illustrated how phase transition behaviour can be used to study heuristics. By means of an annealed theory, we defined a parameter, κ that measures the "constrainedness" of an ensemble of number partitioning problems. Contrary to a conjecture of Y. Fu (Fu, 1989), a phase transition occurs at a critical value of this parameter. Finite size scaling methods describe the shape of this phase transition. We demonstrated that constrainedness and finite size scaling offer a precise and quantitative means for comparing the performance of algorithms and heuristics for number partitioning. In addition, constrainedness can be used to analyse the heuristics themselves. We predict that a similar methodology will be useful in a wide variety of NP-complete problems.

ACKNOWLEDGEMENTS

The second author was supported in part by a HCM Postdoctoral Fellowship and by EPSRC grant GR/L/24014. We thank Alan Bundy, the members of the Mathematical Reasoning Group at Edinburgh, Fausto Giunchiglia and the members of the Mechanized Reasoning Group at Trento for many CPU cycles donated to these and other experiments. We also thank Richard Korf for providing us with his code for the CKK and greedy backtracking algorithms, and for suggesting number partitioning as one of the experimental themes at the First AI and OR Workshop, held at Timberline, Oregon in June 1995. Both authors are members of the Algorithms, Problems and Empirical Studies Group (APES), and we thank the other members of this group at Leeds and Strathclyde for discussion and feedback.

References

Barber, M. N. (1983). Finite-size scaling. In *Phase Transitions and Critical Phenomena, Volume 8*, pp. 145–266. Academic Press.

Cheeseman, P., Kanefsky, B., & Taylor, W. (1991). Where the really hard problems are. In *Proceedings of the 12th IJCAI*, pp. 331–337. International Joint Conference on Artificial Intelligence.

Chvatal, V., & Reed, B. (1992). Mick gets some (the odds are on his side). In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pp. 620–627. IEEE.

Fu, Y. (1989). The uses and abuses of statistical mechanics in computational complexity. In Stein, D. (Ed.), *Lectures in the Sciences of Complexity*, pp. 815–826. Addison-Wesley Longman.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability : a guide to the theory of NP-completeness*. W H Freeman.

Gent, I. P., & Walsh, T. (1994). The hardest random SAT problems. In Nebel, B., & Dreschler-Fischer, L. (Eds.), *KI-94: Advances in Artificial Intelligence. 18th German Annual Conference on Artificial Intelligence*, pp. 355–366. Springer-Verlag.

Gent, I., MacIntyre, E., Prosser, P., B.M.Smith, & Walsh, T. (1996). An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of CP-96*, pp. 179–193. Springer Verlag.

Gent, I., MacIntyre, E., Prosser, P., & Walsh, T. (1995). Scaling effects in the CSP phase transition. In *1st International Conference on Principles and Practices of Constraint Programming (CP-95)*, pp. 70–87. Springer-Verlag.

Gent, I., MacIntyre, E., Prosser, P., & Walsh, T. (1996). The constrainedness of search. In *Proceedings of AAAI-96*, pp. 246–252.

Gent, I., & Walsh, T. (1996a). Phase transitions and annealed theories: Number partitioning as a case study. In *Proceedings of ECAI-96*, pp. 170–174.

Gent, I., & Walsh, T. (1996b). The TSP phase transition. *Artificial Intelligence*, *88*, 349–358.

Hogg, T. (1995). Exploiting problem structure as a search heuristic. Tech. rep., Dynamics of Computation Group, Xerox Palo Alto Research Center. PARC Preprint, ftp://parcftp.xerox.com/pub/dynamics/constraints.html.

Karmarkar, N., & Karp, R. (1982). The differencing method of set partitioning. Technical report UCB/CSD/82-114, Computer Science Division, University of

California, Berkeley, California.

Karmarkar, N., Karp, R., Lueker, J., & Odlyzko, A. (1986). Probabilistic analysis of optimum partitioning. *Journal of Applied Probability*, *23*, 626–645.

Kirkpatrick, S., & Selman, B. (1994). Criticial behaviour in the satisfiability of random boolean expressions. *Science*, *264*, 1297–1301.

Korf, R. (1995). From approximate to optimal solutions: A case study of number partitioning. In *Proceedings of the 14th IJCAI*. International Joint Conference on Artificial Intelligence.

Mitchell, D., Selman, B., & Levesque, H. (1992). Hard and Easy Distributions of SAT Problems. In *Proceedings of the 10th National Conference on AI*, pp. 459–465. American Association for Artificial Intelligence.

Pemberton, J., & Zhang, W. (1996). Epsilon-transformation: exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, *81*, 297–325.

Ruml, W., Ngo, J., Marks, J., & Shieber, S. (1994). Easily searched encodings for number partitioning. Tech. rep. TR-10-94, Center for Research in Computing Technology, Harvard University.

Selman, B., & Kirkpatrick, S. (1996). Critical behavior in the computational cost of satisfiability testing. *Artificial Intelligence*, *81*(1-2), 273–296.

Smith, B., & Dyer, M. (1996). Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, *81*, 155–181.

Williams, C., & Hogg, T. (1994). Exploiting the deep structure of constraint problems. *Artificial Intelligence*, *70*, 73–117.