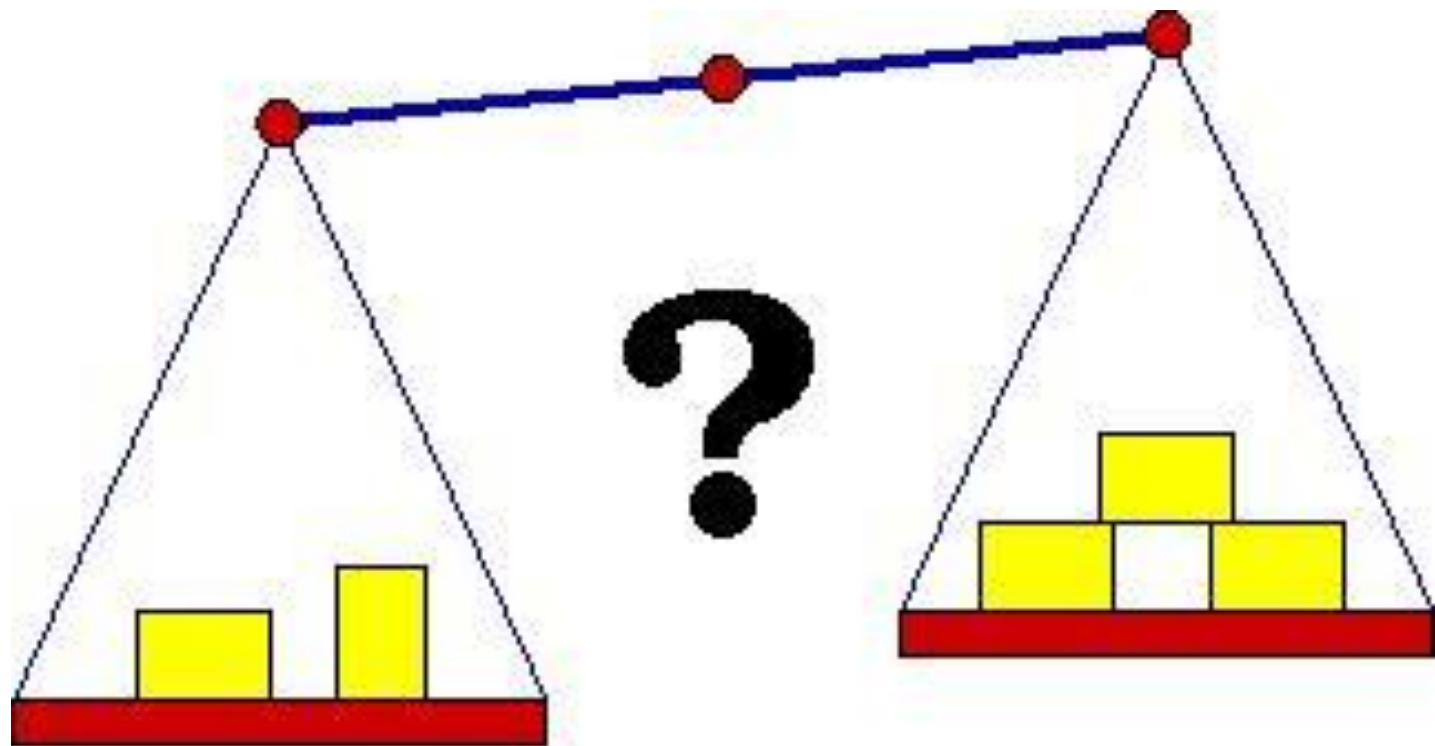


number partitioning



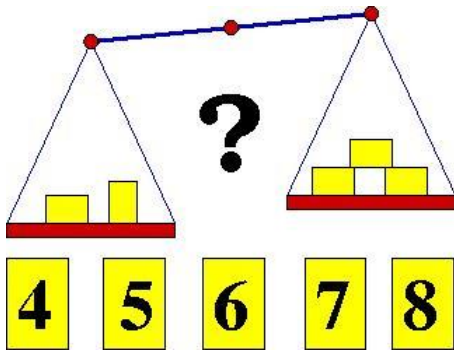
4

5

6

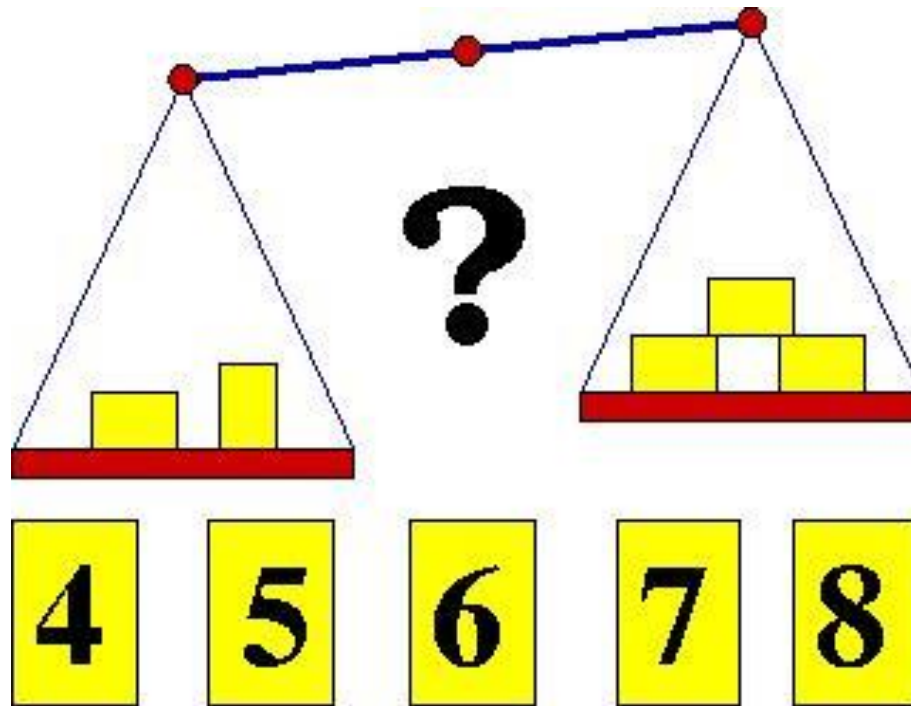
7

8

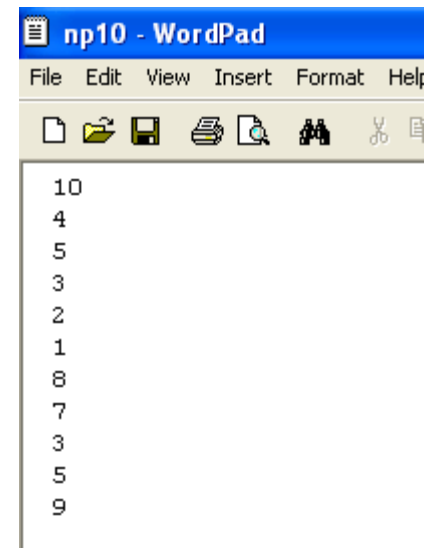


Given a *bag* of numbers, can you partition this into 2 bags such that the sum of the integers in each bag is equal?

Recently featured in the Crystal Maze!
(Thanks Zoe!)

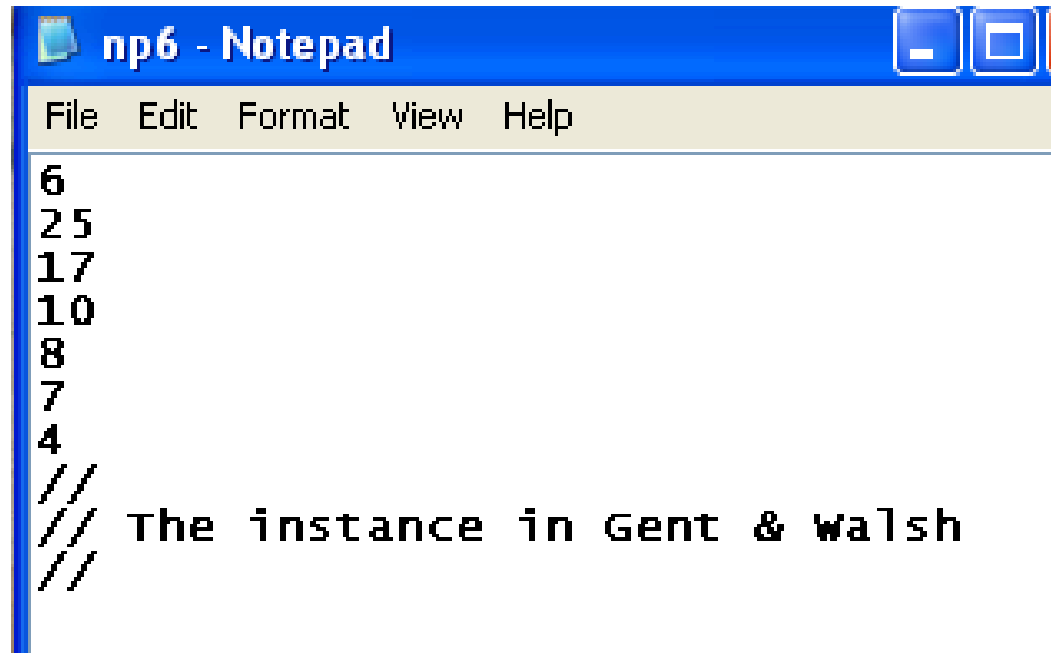


Try it!



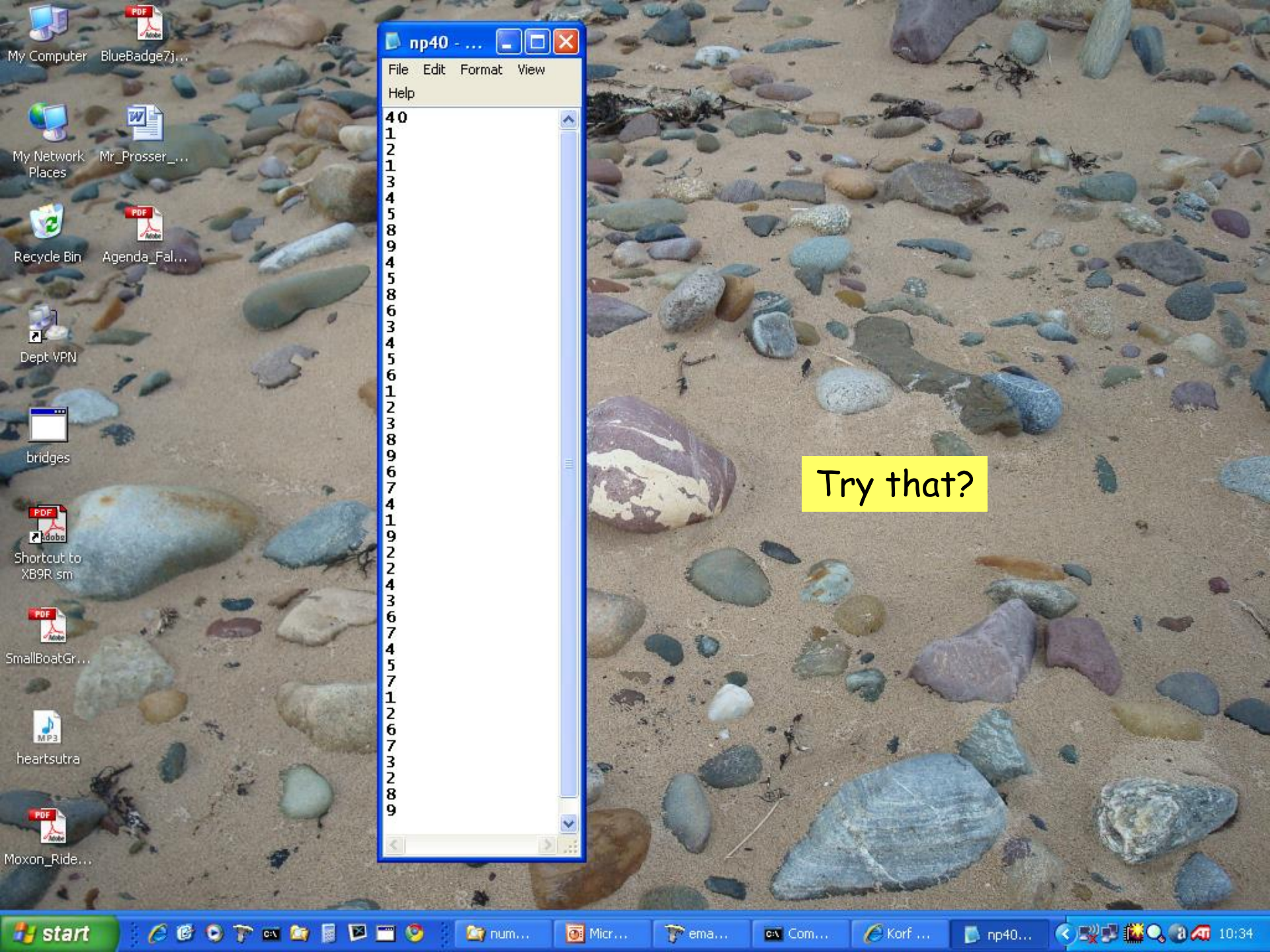
4 5 3 2 1 8 7 3 5 9

Try that!



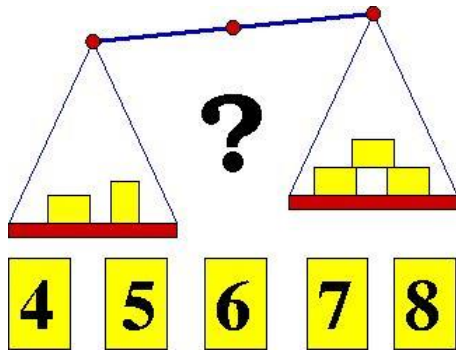
```
6  
25  
17  
10  
8  
7  
4  
///  
/// The instance in Gent & walsh  
///
```

Try that?



```
np40 - ...
File Edit Format View
Help
40
1
2
1
3
4
5
8
9
4
5
8
6
3
4
5
6
1
2
3
8
9
6
7
4
1
9
2
2
4
3
6
7
4
5
7
1
2
6
7
3
2
8
9
```

Try that?



Given a bag of numbers, can you partition this into 2 bags such that the sum of the integers in each bag is equal?

Garey & Johnson "Computers and Intractability"

[SP12] PARTITION

INSTANCE: Finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$

3.2 WEIGHTED SET PROBLEMS

[SP1] PARTITION

INSTANCE: Finite set A and a size $s(a) \in Z^+$ for each $a \in A$.

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A} s(a)/2$?

Reference: [Karp, 1972]. Transformation from 3DM (see Section 3.1.5).

Comment: Remains NP-complete even if we require that $|A'| = |A|/2$, or if the elements in A are ordered as a_1, a_2, \dots, a_n , and we require that A' contain exactly one of a_{2i-1}, a_{2i} for $1 \leq i \leq n$. However, all these problems can be solved in pseudo-polynomial time by dynamic programming (see Section 4.2).

[SP2] SUBSET SUM

INSTANCE: Finite set A , size $s(a) \in Z^+$ for each $a \in A$, positive integer B .

QUESTION: Is there a subset $A' \subseteq A$ such that the sum of the sizes of the elements in A' is exactly B ?

Reference: [Karp, 1972]. Transformation from PARTITION.

Comment: Solvable in pseudo-polynomial time (see Section 4.2).

How complex?

Who cares?

Imagine you have 2 machines on the shop floor
You have n activities, of varying durations
Place the activities on the machines to minimise makespan

Why just 2-partition?
Why not m-way partitioning?
Is there an optimisation problem?

A choco4 model

```
Z:\public_html\cpM\choco4\numPart\Decision.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Decision.java
1 //
2 // Given an instance, is there a partition +/- 1?
3 //
4 import java.io.*;
5 import java.util.*;
6 import org.chocosolver.solver.Model;
7 import org.chocosolver.solver.Solver;
8 import org.chocosolver.solver.variables.IntVar;
9 import org.chocosolver.solver.search.strategy.Search;
10
11 public class Decision {
12
13
14     public static void main(String[] args) throws IOException {
15         Scanner sc = new Scanner(new File(args[0]));
16         int n = sc.nextInt(); // number of numbers
17         Model model = new Model("number partitioning");
18         Solver solver = model.getSolver();
19         IntVar[] D = model.intVarArray("D",n,0,1); // decision ... left or right?
20         int[] w = new int[n]; // weights
21         int total = 0; // sum of weights
22
23         for (int i=0;i<n;i++){
24             w[i] = sc.nextInt();
25             total = total + w[i];
26         }
27         sc.close();
28
29         model.scalar(D,w,"=",total/2).post();
30         //solver.limitTime(10000); // how? that's how
31         solver.setSearch(Search.minDomUBSearch(D)); // take 1 then 0
32
33         boolean solved = solver.solve();
34         System.out.print(solved);
35         System.out.println(" nodes: "+ solver.getMeasures().getNodeCount());
36     }
37 }
```

Java source file length: 1,210 lines: 38 Ln: 2 Col: 48 Sel: 0 | 0 Windows (CR LF) UTF-8 INS

```
Z:\public_html\cpM\choco4\numPart\data\np6 -... - □ ×
File Edit Search View Encoding Language Settings Tools Macro
Run Plugins Window ? X
Decision.java × np6 ×
1 6
2 25
3 17
4 10
5 8
6 7
7 4
8 //
9 // The instance in Gent & Walsh
10 //
Ln: 10 Col: 3 Sel: 0|0 Windows (CR LF) UTF-8 INS
```

```
nauru.dcs.gla.ac.uk - PuTTY
>>
>>
>>
>>
>>
>>
>> javac *.java
>> java Decision data/np6
true nodes: 2
>>
```



```

Scanner sc      = new Scanner(new File(args[0]));
int n          = sc.nextInt(); // number of numbers
Model model    = new Model("number partitioning");
Solver solver  = model.getSolver();
IntVar[] D     = model.intVarArray("D",n,0,1); // decision ... left or right?
int[] w        = new int[n]; // weights
int total      = 0; // sum of weights

for (int i=0;i<n;i++){
    w[i] = sc.nextInt();
    total = total + w[i];
}
sc.close();

```

The screenshot shows a text editor window with the following content:

```

1 6
2 25
3 17
4 10
5 8
6 7
7 4
8 //
9 // The instance in Gent & Walsh
10 //

```

The status bar at the bottom indicates: Ln: 10 Col: 3 Sel: 0|0 Windows (CR LF) UTF-8 INS

```
model.scalar(D,w,"=",total/2).post();  
//solver.limitTime(10000); // how? that's how  
solver.setSearch(Search.minDomUBSearch(D)); // take 1 then 0
```

```
boolean solved = solver.solve();  
System.out.print(solved);  
System.out.println(" nodes: "+ solver.getMeasures().getNodeCount());
```

- Is there a better way to do numPart?

Computational Intelligence, Volume 14, Number 3, 1998

ANALYSIS OF HEURISTICS FOR NUMBER PARTITIONING

IAN P. GENT AND TOBY WALSH

Department of Computer Science, University of Strathclyde, Glasgow

We illustrate the use of phase transition behavior 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 analyze 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 heuristic are superior at reducing constrainedness. This supports the better performance observed experimentally for the Karmarkar–Karp heuristic. Our results refute a conjecture of Fu that phase transition behavior does not occur in number partitioning. Additionally, they demonstrate that phase transition behavior is useful for more than just simple benchmarking. It can, for instance, be used to analyze 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 surprisingly easy to solve. One place to find hard instances is at a phase transition

432 COMPUTATIONAL INTELLIGENCE

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 (1982) replaces two numbers by their difference. 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

head of a binary tree, we search a binary tree, where at each branch the corresponding number is assigned to one of the k subsets. The second pruning rule above is replaced by the following. Let t be the sum of all the numbers, s the current largest subset sum, and d the difference of the best complete partition found so far. If $s - \frac{t-s}{k-1} \geq d$, terminate this branch. The reason is that the best we could do would be to perfectly equalize the remaining $k - 1$ subsets, and if this would result in a partition no better than the best so far, there is no reason to continue searching that path. At each branch, we place the next number in the subsets in increasing order of their sums, to minimize the time to find a good solution.

2.3 Karmarkar-Karp Heuristic (KK)

A heuristic much better than greedy was called set differencing by its authors [Karmarkar and Karp, 1982], but is usually referred to as the KK heuristic. It places the two largest numbers in different subsets, without determining which subset each goes into. This is equivalent to replacing the two numbers with their difference. For example, placing 8 in subset A and 7 in subset B is equivalent to placing their difference of 1 in subset A , since we can always subtract the same amount from both sets without affecting the solution. Swapping their positions is equivalent to placing the 1 in subset B . The KK heuristic repeatedly replaces the two largest numbers with their difference, inserting the new number in the sorted order, until there is only one number left, which is the final partition difference. In our example, this results in the series of sets (8,7,6,5,4), (6,5,4,1), (4,1,1), (3,1), (2). Some additional bookkeeping is required to extract the actual partition, which in this case is (7,5,4) and (8,6), with a partition difference of $16-14=2$. The solution quality of this heuristic is the last remaining number, which is much smaller than the smallest original number, due to the repeated differencing.

2.4 Complete Karmarkar-Karp Algorithm (CKK)

We extended the KK heuristic to the complete Karmarkar-Karp algorithm (CKK) [Korf, 1998]. While the KK heuristic always places the two largest numbers in different subsets, the only other option is to assign them to the same subset. This is done by replacing the two largest numbers by their sum. CKK searches a binary tree where at each node the left branch replaces the two largest numbers by their difference, and the right branch replaces them by their sum. By searching

each triple sorted in decreasing order, and the triples sorted in decreasing order of their largest sum. Initially, each number is in a separate triple, with zero for the remaining numbers. For example, the initial state of a three-way KK partition of the set (4,5,6,7,8) would be ((8,0,0),(7,0,0),(6,0,0),(5,0,0),(4,0,0)). At each step of the KK heuristic, if (a, b, c) and (x, y, z) are the triples with the largest numbers, they are replaced with $(a+z, b+y, c+x)$, which is then normalized by subtracting the smallest element of the triple from each element. This combination is chosen to minimize the largest values. In our example, this results in the set ((8,7,0),(6,0,0),(5,0,0),(4,0,0)). Combining the next two largest triples results in the set, ((8,7,6),(5,0,0),(4,0,0)), which after normalization is represented by ((5,0,0),(4,0,0),(2,1,0)). Combining the next two results in ((5,4,0),(2,1,0)), and combining the last two produces (5,5,2) or (3,3,0) after normalizing, for a final partition difference of 3, corresponding to the partition (8), (7,4), and (6,5), which happens to be optimal in this case.

For the complete CKK algorithm, at each node we combine the two triples with the largest sums in every possible way rather than just one way, branching on each combination. For three-way partitioning, there are six ways to combine two triples, corresponding to different permutations of three elements, and for k -way partitioning, there are $k!$ branches at each node. For example, given the triples (a,b,c) and (x,y,z) , their different possible combinations are $(a+x, b+y, c+z)$, $(a+x, b+z, c+y)$, $(a+y, b+x, c+z)$, $(a+y, b+z, c+x)$, $(a+z, b+x, c+y)$, and $(a+z, b+y, c+x)$. Since the smallest sum of each k -tuple is always zero after normalization, we only maintain tuples of $k - 1$ sums for k -way partitioning.

2.5 Pseudo-Polynomial-Time Algorithms

Technically, number partitioning is not strongly NP-complete, but can be solved in pseudo-polynomial-time by dynamic programming. This requires memory that is proportional to $n(k-1) \cdot m^{k-1}$ for k -way partitioning of n numbers with a maximum value of m . As a result, these algorithms are not practical for multi-way partitioning. For example, three-way partitioning of 40 7-digit integers requires a petabyte (10^{15}) of storage.

2.6 Previous State-Of-The-Art

For two-way partitioning, CKK is the algorithm of choice. It is slightly faster than CGA without perfect partitions, and

International Joint Conference on Artificial Intelligence
Twenty-First International Joint Conference on Artificial Intelligence

PRESENTATIONS

Reading Tools

Multi-Way Number P...

Korf

- Abstract
- Review policy
- About the author
- How to cite item
- Indexing metadata
- Print version
- Notify colleague*
- Email the author* Add comment†

SEARCH CONFERENCE

All

... and now for something completely different!!!



Assume for sake of argument, we have 3 digit numbers

Will it be easier to partition a bag of 100 numbers or a bag of 10 numbers?

Will we always be able to partition a bag of numbers?

Random Data

```
Z:\public_html\cpM\choco4\numPart\RandomGen1.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Decision.java x np6 x RandomGen1.java x
1  import java.util.Random;
2
3  public class RandomGen1 {
4
5      public static void main(String[] args) {
6          int n      = Integer.parseInt(args[0]); // number of numbers
7          int d      = Integer.parseInt(args[1]); // number of digits
8          long x     = 0;
9          Random gen = new Random();
10
11         System.out.println(n);
12         for (int i=0;i<n;i++){
13             x = 0;
14             for (int j=0;j<d;j++) x = x * 10 + gen.nextInt(10);
15             System.out.println(x + " ");
16         }
17     }
18 }
19
Java source length : 447 lines : 19 Ln : 1 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS
```

Random Data

```
simeulue.dcs.gla.ac.uk - PuTTY
>> javac *.java
>> java RandomGen1 20 6
20
736293
227546
623540
130944
341470
330426
471525
503542
210678
359409
217011
82106
5846
89623
662342
31974
465222
221892
922361
747946
>>
```

Answer these questions

1. As we increase n does the problem get easier?
2. As we increase n do more or less instances have partitions?
3. As we increase d do problems get easier or harder?

Experiment 1: $d = 3$, vary n from 9 to 26 in single steps, sample size 10

Experiment 2: $d = 3$, vary n from 100 to 500 in steps of 100, sample size 10

Experiment 3: $d = 6$, vary n from 15 to 26 in single steps, sample size 10

Experiment 4: $d = 6$, vary n from 100 to 500 in steps of 100, sample size 10

Experiment 5: $d = 7$, $n = 100$, sample size 10

Observe % solubility and search effort