The Partition Problem, and How The Distribution of Input Bits Affects the Solving Process

Nikita Sazhinov[1][a], Ruben Horn[1,2][b], Pieter Adriaans[3,4][c] and Daan van den Berg[1,3][d]

[1]*Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands*

[2]*Helmut-Schmidt-University Hamburg, Germany*

[3]*University of Amsterdam, The Netherlands*

[4]*Institute for Logic, Language, and Computation, The Netherlands*

Keywords: Partition Problem, Subset Sum, Instance Hardness, Exact Algorithms, Heuristic Algorithms, Karmarkar-Karp.

Abstract: The hardness of the partition problem does not only depend on the number of integers in the problem instance, but also on their magnitude, measured in informational bits. In this work, we will show that also the exact *distribution* of informational bits among the integers influences the hardness for at least one exact and three heuristic algorithms.

1 INTRODUCTION

"The easiest hard problem" (Mertens, 2006; Hayes, 2002) must be the best nickname ever given to an NP-hard problem. It belongs to the well-known *partition problem* (Korf, 1998), which in its most common form, involves splitting a set of n integers in two, so that their summed values are as close as possible. For instance, the set

$$S_1 = \{242, 238, 181, 165, 134, 208, 161, 181\} \quad (1)$$

can be split in {238, 181, 134, 208} and {242, 165, 161, 181} which sum up to 761 and 749 respectively and hence have a difference of 12. It has another solution, when split in subsets {242, 165, 134, 208} and {238, 181, 161, 181}, which *also* have a difference of 12, but a better solution, meaning a closer split with a smaller difference, does not exist. This instance therefore has multiple global minima, but no perfect splits. Contrarily, set

$$S_2 = \{207, 153, 186, 146, 230, 217, 175, 212\} \quad (2)$$

has only one solution and it is *perfect*, meaning the difference between the subsets is zero – {207, 153, 186, 217} and {146, 230, 175, 212}, both summing up to 763. Finding a perfect solution during the search process is great, because it allows the algorithm to halt immediately, saving computational costs.

This property, the a priorily known optimum of zero for all imaginable instances, sets the partition problem apart from other NP-hard problems. In the traveling salesman problem for example, the value of the optimal solution of some given instance cannot be known aforehand, with the exception of a few rare cases. This prior knowledge greatly influences the solve time for a partition problem instance; the more perfect solutions a problem instance has, the sooner one will be found, and the earlier the search can be halted. As it turns out, this number of perfect solutions depends on m/n, in which m is the number of informational bits of a typical integer of the set (Figure 1). The higher m/n, the lower the number of perfect solutions. In other words: if the instance's integers are small compared to its number of integers, the number of perfect solutions is high, and the instance is expected to be relatively easy. If on the other hand, a same-sized instance holds *large* integers, it is significantly harder because it has few or no perfect solutions, thereby requiring a long time to find the optimal split, much like solving a traveling salesman problem. The 8-integer instance S_3 for example

[a] https://orcid.org/0000-0002-2089-9602
[b] https://orcid.org/0000-0001-6643-5582
[c] https://orcid.org/0000-0002-8473-7856
[d] https://orcid.org/0000-0001-5060-3342

Sazhinov, N., Horn, R., Adriaans, P. and van den Berg, D.
The Partition Problem, and How The Distribution of Input Bits Affects the Solving Process.
DOI: 10.5220/0012143600003595
In *Proceedings of the 15th International Joint Conference on Computational Intelligence (IJCCI 2023)*, pages 143-150
ISBN: 978-989-758-674-3; ISSN: 2184-3236

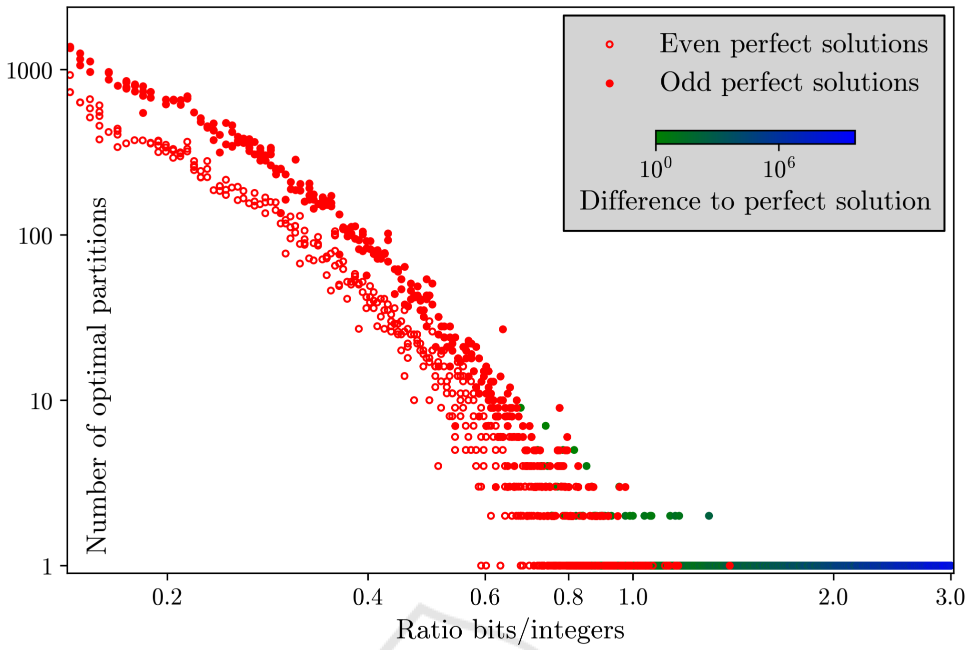

Figure 1: For the partition problem, the number of ways to reach the optimal solutions depends on both the number of integers n and informational bits m. As the ratio m/n exceeds 1, the optimal solution is likely unique, making the problem much harder.

$$S_3 = \{6, 8, 4, 12, 2, 10, 14, 16\} \tag{3}$$

is very easy. Its perfect split is 36-36, and there are many ways to achieve one[1]. Set S_4 on the other hand, also holding 8 integers

$$S_4 = \{76579, 62450, 91942, 9188, 68382, 30234, 36504, 28103\} \tag{4}$$

has no perfect split, so an exact algorithm *has* to search the entire search tree to guarantee that it finds the best solution, at great computational cost. This dependency, the dependency of the computational hardness not only on the *number* of variables in an instance but also on their *magnitude*, measured in informational bits, is know as 'weak NP-hardness' (Hayes, 2002).

In this study, we go one step beyond n and m in analyzing the hardness of the partition problem. We will generate 7000 partition problem instances of equal size, all $n = 14$ integers and $m = 105$ informational bits, but we will vary the *distribution* of bits over the integers. Using 7 different distributions ("strict templates", see Table 1) to generate 1000 random instances each, we will solve these with one exact algorithm and three heuristic algorithms and evaluate the results.

About the organization of this paper: the next section can be safely skipped for those already familiar in the field; it provides related work on the partition problem and about easy-hard phase transitions in other problems. Section 3 details on the generation of the 7000 problem instances and their bit distributions, while Section 4 explains the algorithms used to solve these. The effects of the bit distributions on the efficiency of the algorithms are presented in the results (Section 5) after which the conclusion and discussion in Section 6 wrap up the read.

2 RELATED WORK

The partition problem can be seen as a special case of the multi-way partition problem, where rather than dividing the set into two equal sum subsets, one divides it into k subsets (Martens, 2006) and finds applications in areas such as task scheduling (Graham, 1969), voting manipulation (Walsh, 2009), multiprocessor scheduling (Schreiber et al., 2018; Dell'Amico and Martello, 1995; Dell'Amico et al., 2008). The most prominent name in this field is probably that of Richard Korf (Korf, 2009; Schreiber et al., 2018), who published for over two decades on algorithmics for the (multi-way) partitioning problem, e.g. improving on the Karmarkar-Karp algorithm (Karmarkar and

[1]Try it yourself.

Table 1: The 7 strict templates used in this experiment, each with an example partition problem instance. All generated instances have exactly 14 integers and exactly 105 informational bits, but bits are distributed differently over the integers for each template.

	Strict Template	**Example instance**
ST_3	26b, 20b, 17b, 13b, 9b, 4b, 3b, 3b, 3b, 3b, 2b, 2b, 1b, 1b	38553645, 832461, 111689, 4981, 357, 14, 7, 6, 5, 4, 3, 2, 1, 0
ST_2	22b, 18b, 15b, 12b, 9b, 6b, 5b, 5b, 4b, 3b, 2b, 2b, 1b, 1b	2614847, 158808, 24310, 2818, 511, 59, 24, 23, 9, 5, 3, 2, 1, 0
ST_1	17b, 15b, 13b, 11b, 10b, 9b, 7b, 6b, 5b, 4b, 4b, 2b, 1b, 1b	120760, 32272, 8143, 2011, 591, 472, 77, 46, 21, 15, 7, 3, 1, 0
ST_0	14b, 13b, 12b, 11b, 10b, 9b, 8b, 7b, 6b, 5b, 4b, 3b, 2b, 1b	15411, 6345, 3880, 1947, 783, 469, 202, 110, 40, 16, 13, 3, 1, 0
ST_{-1}	13b, 12b, 11b, 10b, 9b, 9b, 9b, 7b, 6b, 5b, 4b, 4b, 3b, 3b	4272, 2169, 1294, 682, 440, 316, 276, 66, 52, 31, 10, 8, 6, 5
ST_{-2}	10b, 10b, 10b, 8b, 8b, 8b, 7b, 7b, 7b, 6b, 6b, 6b, 6b, 6b	808, 703, 564, 221, 188, 133, 122, 86, 72, 63, 59, 53, 46, 40
ST_{-3}	8b, 8b, 8b, 8b, 8b, 8b, 8b, 7b, 7b, 7b, 7b, 7b, 7b, 7b	251, 246, 229, 225, 198, 166, 146, 118, 116, 109, 93, 89, 84, 81

Karp, 1982; Korf, 1995; Korf, 1998).

It's actually remarkable that Korf has (apparently) never attempted to quantify instance hardness for the partition problem. His earlier work on the asymmetric traveling salesman problem, together with WeiXiong Zhang, yielded a tremendous insights on how the performance of an exact algorithm's performance depends on the actual numerical values inside the distance matrix (Zhang and Korf, 1996). We cannot escape the feeling that constrainedly summing up integers from a matrix such as in asymmetric traveling salesman problem is very close to the partition problem, or at least to subset sum.

Zhang & Korf's work was at least partially inspired by an earlier work on instance hardness for ATSP, namely the investigation of Cheeseman, Kanefsky and Taylor[2] (Cheeseman et al., 1991). It must be the most cited paper in the field, showing that for ATSP, the standard deviation of the integers in the cost matrix of an instance was the critical indicator of its algorithmic hardness. Sadly, Cheeseman et al. were wrong, as they likely overlooked a roundoff error which was only discovered three decades later, practically nullifying all these results (Sleegers et al., 2020). But that's science, moving ahead by a stride and a stumble. Still, Cheeseman et al.'s paper inflamed the instance hardness discussion like no other.

As once noted by Kevin Leighton-Brown, instance hardness appears to be much better investigated on NP-complete decision problems (Leyton-Brown et al., 2002). At the root of all decision problems sits satisfiability, for which a solvability phase transition, and accompanying instance hardness was identified in the number of clauses over the number of variables (Larrabee and Tsuji, 1992; Kirkpatrick and Selman, 1994; Gent and Walsh, 1994). The Hamiltonian cycle problem too, has a solvability phase transition, through its edge degree, as was demonstrated by Cheeseman et al. in the same paper as the ATSP-hardness (Cheeseman et al., 1991). This experiment was later independently verified in an extended replication by Joeri Sleegers (van Horn et al., 2018). But Sleegers moved beyond edge degree, showing not only these results were valid for *all* major exact algorithms (Sleegers and van den Berg, 2021; Sleegers et al., 2022), but also that the hardest existable instances were in a completely different region of the combinatorial state space (Sleegers and van den Berg, 2020a; Sleegers and van den Berg, 2020b; Sleegers and van den Berg, 2022). The fact that these instances do not turn up in random ensembles might be due to their high degree of (Kolmogorov-)structure, and has direct implications for our benchmarking practices (Bartz-Beielstein et al., 2020).

For the partition problem however, recent work on instance hardness showed that the distribution of informational bits among the integers play a critical role in instance hardness[3] (van den Berg and Adriaans, 2021). Their work is rooted in information theory (Adriaans, 2021), but their experiment, though interesting, is fairly small. In our experiments, we will add three more algorithms, and 100-fold the number of instances, which we will discuss in the following section.

3 PROBLEM INSTANCES

For generating partition problem instances, we use 'strict templates' containing explicit designations for informational bits to integers. For example, a strict template like $(4b, 3b, 2b, 1b)$ could give rise to randomly generated instance such as $\{14, 5, 3, 1\}$ or $\{9, 6, 2, 0\}$. Here, the first integer containing exactly 4 bits of information, the second containing exactly 3 bits, then 2 bits and finally 1 bit. Note that the value 0 is a borderline case here, as it is discardable as an input integer for the partition problem, but we leave it in for generative purposes. A second remark contains the notion of 'strict', meaning that e.g. a 4-bit

[2]One can tell not only from their references, but also their reported personal communication with Peter Cheeseman.

[3]The authors actually use the more general term 'subset sum', but are practically investigating the partition problem.

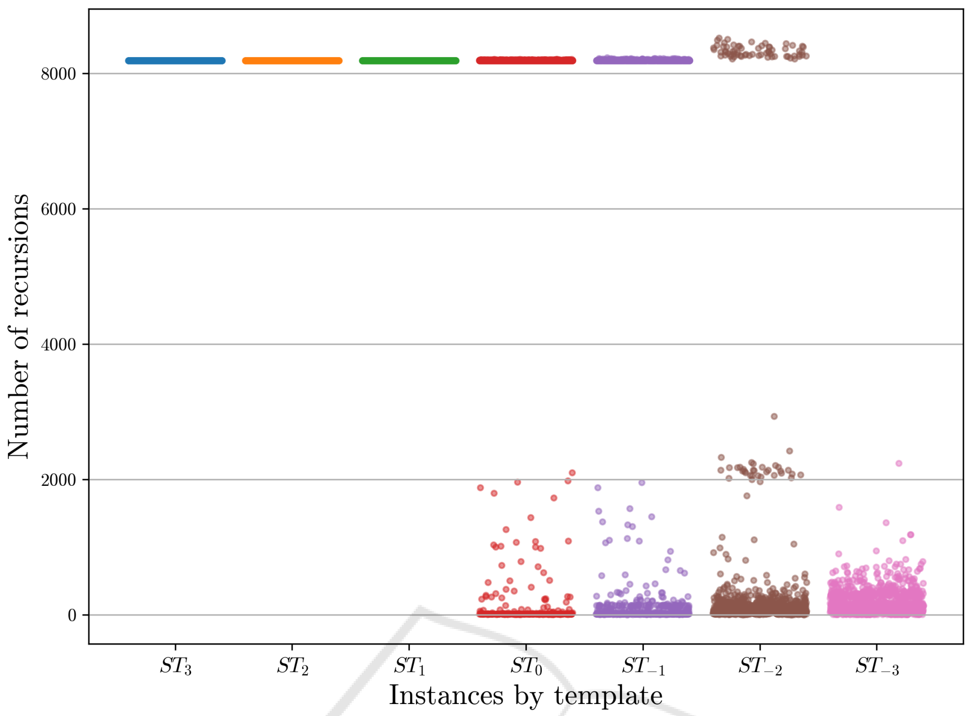

Figure 2: Recursions required for the branch and bound algorithm on all 7000 partition problem instances. Different strict templates representing different bit distributions are represented by different colours, and show the impact on the instances' hardness.

number has exactly 4 bits of information, and no less. Therefore, integers valued $8 \leq x \leq 15$ are considered strict 4-bit numbers, as smaller numbers can also be represented with fewer bits of information. This notion is needed pertaining the weak NP-hardness of the partition problem: we want to compare instances with identical numbers of integers *and* identical total numbers of informational bits.

The strict templates (ST) we used all generated instances of 14 integers with a total of 105 informational bits. The central and in some ways most regular strict template is ST_0, which is linearly decreasing in its informational bits: {14b, 13b ,12b ... 1b} 1. Going down through ST_{-1} ST_{-2} and ST_{-3}, templates are becoming 'flatter', with the number of bits more equally distributed among the integers; ST_{-3} is the flattest template possible for these numbers of integers and bits. Going up through ST_1 ST_2 and ST_3, templates become ever more 'eccentric', with steep differences between the largest and the smallest entries. Despite being on top, ST_3 is certainly not the most eccentric template imaginable, but preliminary investigations and postliminary reasoning affirmed that strict templates above ST_3 will not induce substantially different algorithmic behaviour over ST_3. It should be noted that our templates are a bit different from Van den Berg & Adriaans's work (van den Berg and Adriaans, 2021), who accidentally generated multisets along their regular sets (Adriaans and van den Berg, 2023). Even though we do not expect the experimental results to be substantially different when allowing multisets, we prefer to stay closer to the most common definition of the partition problem.

From each of the 7 strict templates, we made 1000 randomized partition problem instances, totalling to 7000 problem instances of 14 integers, all of which are unique. This is 100 times more than the early study on this subject, exposing a lot of fine grain in the hardness diagram. Apart from that, we will investigate the effect of these template-based instances on the performance of some common heuristic algorithms for the partition problem. All instances, codes, figures and supplementary material can be found in an online repository (Horn, 2022).

4 ALGORITHMS

The 7000 instances genererated by the 7 strict templates are all solved by 1 exact algorithm and 3 heuristic algorithms, all of which require the problem instance to be sorted in descending order. Apart from this commonality, there is no real reason why specifically *these* algorithms are used in this study other

Table 2: The performance of the heuristic algorithms, expressed in heuristicDeficiency, for different strict templates used in this study.

Template	**Greedy**			**KK-largdif**			**KK-pairdif**		
-	min	μ	max	min	μ	max	min	μ	max
ST_3	8.231	15.149	26.810	8.231	15.149	26.810	8.275	15.260	27.083
ST_2	2.542	4.361	7.284	2.542	4.361	7.284	2.578	4.427	7.469
ST_1	1.145	1.513	2.114	1.145	1.513	2.114	1.209	1.615	2.255
ST_0	1.000	1.026	1.182	1.000	1.026	1.182	1.000	1.200	1.564
ST_-1	1.000	1.015	1.126	1.000	1.014	1.126	1.000	1.203	1.572
ST_-2	1.000	1.005	1.017	1.000	1.001	1.010	1.000	1.086	1.214
ST_-3	1.000	1.005	1.034	1.000	1.001	1.024	1.000	1.006	1.036
All	1.000	3.582	26.810	1.000	3.581	26.810	1.000	3.685	27.083

than just expanding our knowledge on the subject. We could equally well imagine a similar study being done with dynamic programming, simulated annealing, or the plant propagation algorithm, just to name a few.

The exact algorithm for splitting sets is a depth-first branch and bound algorithm, and our implementation can be seen as binarily looping through the sorted integers of the instance. Initially, the first (and therefore largest) integer is marked as ‘included’. If the included integers sum up to at least the target value of half the summed set, the addition of further integers need not be considered. This value can therefore be considered the ‘bound’ along which the search tree is pruned. Note that this bound thereby is a positive bound: the algorithm should keep branching as long as the incumbent value is *smaller* than the target value, because the closest approximation of the perfect partition might be slightly higher than the target value. After that, the branch *without* the respective integer is considered, adding the second, third and fourth solution to the sum, again backtracking when the target value is exceeded. A particular subset hitting the target value *exactly* constitutes a hard stopping criterion; in this case, it found a perfect partition, and the search can be halted because no better solutions exist. Consequently, when an instance has many perfect partitions, one can be discovered early, halting the search and significantly saving computational costs.

The complete search tree of a problem instance has 2^n nodes, a theoretical upper bound that cannot be reached by branch and bound in this setting of the partition problem, where the target value is always $\frac{1}{2}\sum_{i=1}^{n}(integer_i)$. More general instances of the subset sum problem might make close approximations, but branch and bound’s runtime will be closer to $2^{(n/2)}$ for these particular sorted problem instances. However, as typical for exact algorithms, we will be getting a guaranteed optimal partition for this exponential runtime.

Computational history however, traditionally not too fond of the exponential runtimes that come with solving NP-hard problems, produced a good lot of alternatives. The $O(n)$ greedy heuristic, the Karmarkar-Karp largest difference (KK-largdif) and Karmarkar-Karp paired difference (KK-pairdif) algorithms, both $O(n^2)$, produce good or even near-optimal results for partition problem instances (Hayes, 2002; Karmarkar and Karp, 1982). Although these old algorithms are heuristics, they are deterministic and parameterless; this in stark contrast to many of the bio-inspired population-based algorithms that have been published in the 21^{st} century, often taking myriads of input parameters, and being stochastic in nature. The case for KK-largdif is even stronger; although called a heuristic, it is much better, *guaranteeing* a $7/6 \cdot opt$ solution from the best possible split *opt* in polynomial time, thereby being a 7/6 - approximation algorithm (Fischetti and Martello, 1987).

The greedy algorithm starts off with two empty sets, and iterates through the instance’s sorted integers, assigning each integer to the set with the smallest sum so far. The KK-largdif grabs the first two integers of the sorted instance, which are the two largest, and replaces them by their absolute difference, after which the instance is resorted. The process is repeated until the instance is empty, the last integer is the difference of the split, and the explicit solution can be reconstructed through backtracking if needed.

The KK-pairdif works slightly different, first creating an empty set E_1, which is filled with the difference between the first and second integers from the sorted instance, the third and fourth integers, fifth and sixth, and so forth. E_1 is then sorted, and the operation is repeated on E_1, producing E_2. The process continues until there is only one integer left in E_n, which is the difference of the split, and the explicit solution can be reconstructed through backtracking.

The tradeoff here is clear: a good, but not perfect heuristic solution can be obtained in exchange for better runtimes, but how would the distribution of informational bits affect their performance? We will compare the deficiency of the heuristic algorithms, which

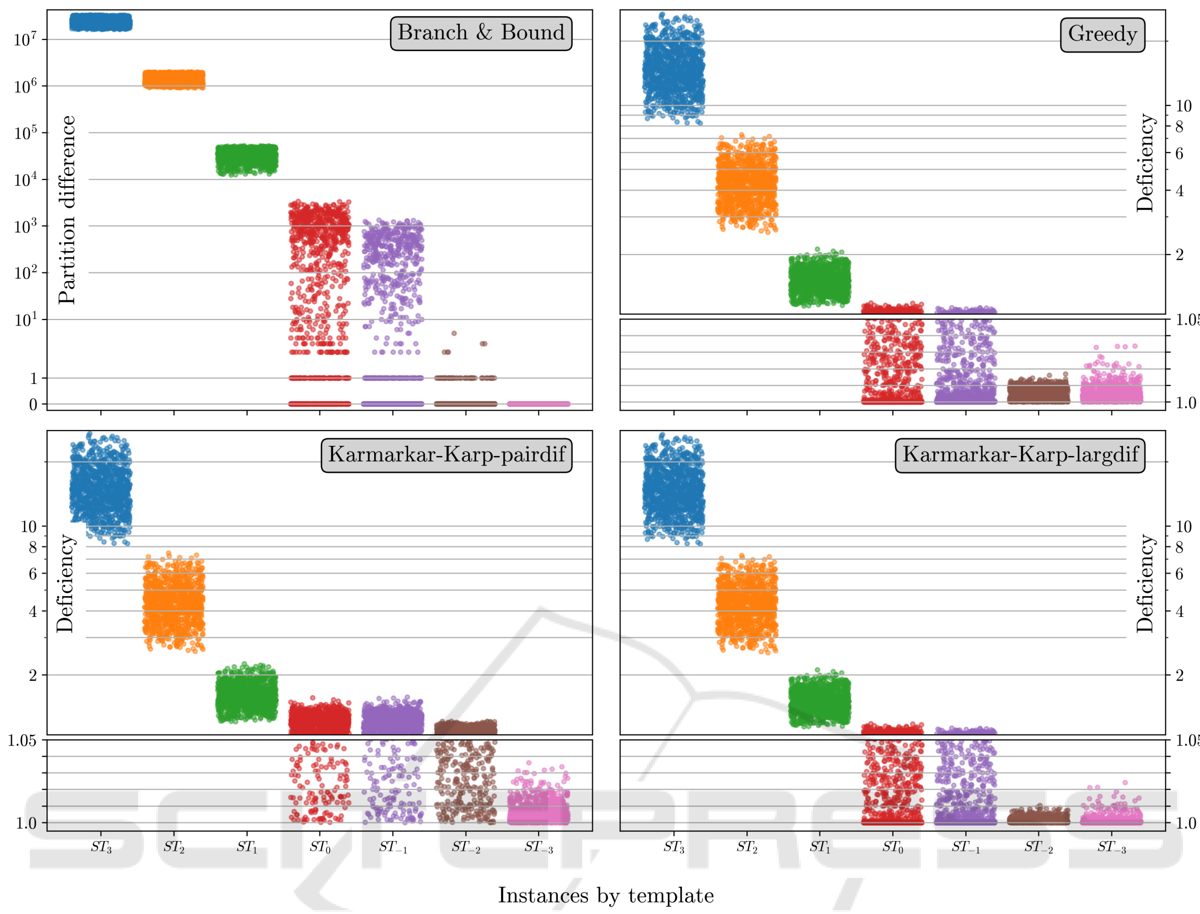

Figure 3: The heuristic performance of the three most popular partition problem heuristics (Greedy, Karmarkar-Karp-largestDifference, Karmarkar-Karp-pairedDifference) is critically influenced by the bit distribution within the instance. Note that the vertical axes denotes the heuristicDeficiency. Top-left subfigure shows the difference between the subsets in the optimal split for instances in a strict template.

is given as

$$heuristicDeficiency = \frac{worstSplit - bestSplit}{worstSplit - heuristicSplit} \quad (5)$$

in which *worstSplit* is the summed instance, *bestSplit* is the best possible split value, guaranteed by the branch and bound algorithm which is an exact algorithm, and *heuristicSplit* is the solution as returned by the respective heuristic algorithm. If the heuristic performs optimal, finding the smallest-difference split, the heuristic deficiency is 1, if it approaches the optimal split only half as good as the exact algorithm, the heuristic deficiency is 2 (read: "it performs twice as bad as the exact algorthm.").

All 28,000 runs were executed on a 2015 HPx360 laptop with an I7 core, and 8GB of memory using single threaded Python (find source code here: (Horn, 2022)). Even on this modest setup, the whole experiment can be completed in a few days, especially when screen logs are turned off. One should remember though, that every extra integer added to the template *doubles* the expected computational effort.

5 RESULTS

The runtimes for branch-and-bound on instances from the eccentric templates ST_3, ST_2 and ST_1 are completely uniform at 8192 recursions. In retrospect, this is no surprise because the first integer is more than one bit larger than the second integer in the set, thereby inevitably constituting more than half of the instance's total sum. Therefore, the entire subtree containing the first integer is immediately discarded, while the entire subtree without the first integer gets checked for a closer approximation of the perfect split, summing up to 2^{n-1} recursions, which is 8192 for our instances of $n = 14$.

The first hardness variations occur in ST_0 and ST_{-1}, which have both higher and lower numbers of recursions (r) with $r \in [5, 8210]$, $\mu = 6262$, $\sigma = 3443$ for ST_0 and $r \in [0, 8235], \mu = 4709, \sigma = 4009$ for ST_{-1} respectively, in which μ is the mean and σ the standard deviation of r. This includes one extraordinary 'rogue' instance in ST_{-1} that required exactly 0 recursions. It is the unlikely but true case of a random instance whose first integer is *exactly* half of the summed set.

ST_{-2} is the template that has generated the hardest instances ($\in [1, 8527], \mu = 722, \sigma = 2061$). It has roughly three hardness groups: one slightly over 8192 recursions, one near 0 recursions, and a very small one just over 2000 recursions. We do not know why this particular hardness grouping occurs, or what sets the particularly hard instances apart.

ST_{-3} produces exclusively easier instances ($\in [4, 2243], \mu = 195, \sigma = 127$) with 99% of its instances under 2000 recursions. This is likely due to the high number of solutions present in instances, such as in S_3 of Example 3.

A few notable differences occurred in the heuristics' performance over the entire ensemble of 7000 instances (Figure 3). When assessing the aggregate values (Table 2), the greedy algorithm and the KK-largdif performed nearly identical. Only 5 out 21 values (max, min, average heuristicDeficiency for 7 templates) differed, and these differences were all smaller than 1%. We expected the difference to be greater, and these results do raise the question for what (kind of) instances these algorithms *do* show substantially different performance.

These results do suggest however, why the second variation of the Karmarkar-Karp algorithm has never gained any popularity. KK-pairdif lagged behind substantially on 17 out of 21 measurements, underperforming from between 0.5% to 39.6%, over half of which was over 5%. – quite significant if we realize that these instances have only 14 integers. It would be interesting to actively seek for which (kinds of) instances these underperformances are maximal. The common denominator for now, is that the performance difference for the heuristic algorithms is largest in strict templates ranging from ST_0 to ST_{-3}, where we also find the largest variations in computational costs for the exact algorithm.

6 CONCLUSION & DISCUSSION

In retrospect, the results are almost trivial: how could the distribution of bits *not* have influenced the hardness? Still, these results paint a different picture from the traditional (weak NP-)hardness of the partition problem. Would it make sense to in some way combine the classical diagram of Figure 1 with Figure 2? It would require a systematic way of quantifying eccentricity in the strict templates, which is not trivial. Maybe just studying non-eccentricity is sufficient, as only the strict templates below ST_0 show interesting behaviour.

A more poignant question is what the hardest instances in ST_{-2} actually look like, and whether any form of universal hardness can be extracted from them. If we generate more instances from this template, how hard can they possibly get? A third avenue of exploration could be the performance of non-deterministic (meta)heuristics such as simulated annealing (Paauw and van den Berg, 2019; Dijkzeul et al., 2022; Dahmani et al., 2020) or the plant propagation algorithm (Vrielink and van den Berg, 2021a; Vrielink and van den Berg, 2021b; De Jonge and van den Berg, 2020). Can strict template generation also influence the structural properties of these algorithms' hardness landscapes?

Many issues are still open, and we cannot wait to start future work, which will surely incorporate larger templates, larger instance ensembles and more detailed hardness pictures.

REFERENCES

Adriaans, P. (2021). Differential information theory. *arXiv preprint arXiv:2111.04335*.

Adriaans, P. and van den Berg, D. (2023). Communication. From personal email communication with Daan van den Berg and Pieter Adriaans.

Bartz-Beielstein, T., Doerr, C., van den Berg, D., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., La Cava, W., Lopez-Ibanez, M., et al. (2020). Benchmarking in optimization: Best practice and open issues. *arXiv preprint arXiv:2007.03488*.

Cheeseman, P. C., Kanefsky, B., Taylor, W. M., et al. (1991). Where the really hard problems are. In *Ijcai*, volume 91, pages 331–337.

Dahmani, R., Boogmans, S., Meijs, A., and van den Berg, D. (2020). Paintings-from-polygons: simulated annealing. In *International Conference on Computational Creativity (ICCC 2020)*.

De Jonge, M. and van den Berg, D. (2020). Parameter sensitivity patterns in the plant propagation algorithm. In *IJCCI*, pages 92–99.

Dell'Amico, M., Iori, M., Martello, S., and Monaci, M. (2008). Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*, 20(3):333–344.

Dell'Amico, M. and Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7(2):191–200.

Dijkzeul, D., Brouwer, N., Pijning, I., Koppenhol, L., and van den Berg, D. (2022). Painting with evolutionary algorithms. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 52–67. Springer.

Fischetti, M. and Martello, S. (1987). Worst-case analysis of the differencing method for the partition problem. *Mathematical Programming*, 37(1):117–120.

Gent, I. P. and Walsh, T. (1994). The sat phase transition. In *ECAI*, volume 94, pages 105–109. PITMAN.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429.

Hayes, B. (2002). Computing science: The easiest hard problem. *American Scientist*, 90(2):113–117.

Horn, R. (2022). Repository containing source material: https://anonymous.4open.science/r/subset-sum-problem-bit-distribution-4FDB/README.md.

Karmarkar, N. and Karp, R. M. (1982). *The differencing method of set partitioning*. Computer Science Division (EECS), University of California Berkeley.

Kirkpatrick, S. and Selman, B. (1994). Critical behavior in the satisfiability of random boolean expressions. *Science*, 264(5163):1297–1301.

Korf, R. E. (1995). From approximate to optimal solutions: A case study of number partitioning. page 266–272.

Korf, R. E. (1998). A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106(2):181–203.

Korf, R. E. (2009). Multi-way number partitioning. *IJCAI*.

Larrabee, T. and Tsuji, Y. (1992). *Evidence for a satisfiability threshold for random 3CNF formulas*. University of California, Santa Cruz, Computer Research Laboratory.

Leyton-Brown, K., Nudelman, E., and Shoham, Y. (2002). Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *International conference on principles and practice of constraint programming*, pages 556–572. Springer.

Martens, S. (2006). The easiest hard problem: Number partitioning. *Computational complexity and statistical physics*, page 125.

Mertens, S. (2006). Number partitioning. *Computational Complexity and Statistical Physics*, page 125.

Paauw, M. and van den Berg, D. (2019). Paintings, polygons and plant propagation. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 84–97. Springer.

Schreiber, E. L., Korf, R. E., and Moffitt, M. D. (2018). Optimal multi-way number partitioning. *Journal of the ACM (JACM)*, 65(4):1–61.

Sleegers, J., Olij, R., van Horn, G., and van den Berg, D. (2020). Where the really hard problems aren't. *Operations Research Perspectives*, 7:100160.

Sleegers, J., Thomson, S. L., and van den Berg, D. (2022). Universally hard hamiltonian cycle problem instances. In *ECTA 2022: 14th International Conference on Evolutionary Computation Theory and Applications*, pages 105–111. SCITEPRESS–Science and Technology Publications.

Sleegers, J. and van den Berg, D. (2020a). Looking for the hardest hamiltonian cycle problem instances. In *IJCCI*, pages 40–48.

Sleegers, J. and van den Berg, D. (2020b). Plant propagation & hard hamiltonian graphs. *Evo* LBA's*, pages 10–13.

Sleegers, J. and van den Berg, D. (2021). Backtracking (the) algorithms on the hamiltonian cycle problem. *arXiv preprint arXiv:2107.00314*.

Sleegers, J. and van den Berg, D. (2022). The hardest hamiltonian cycle problem instances: the plateau of yes and the cliff of no. *SN Computer Science*, 3(5):1–16.

van den Berg, D. and Adriaans, P. (2021). Subset sum and the distribution of information. pages 134–140.

van Horn, G., Olij, R., Sleegers, J., and van den Berg, D. (2018). A predictive data analytic for the hardness of hamiltonian cycle problem instances. *Data Analytics*, 2018:101.

Vrielink, W. and van den Berg, D. (2021a). A dynamic parameter for the plant propagation algorithm. *Evo* LBA's*, pages 5–9.

Vrielink, W. and van den Berg, D. (2021b). Parameter control for the plant propagation algorithm. *Evo* LBA's*, pages 1–4.

Walsh, T. (2009). Where are the really hard manipulation problems? the phase transition in manipulating the veto rule. *arXiv preprint arXiv:0905.3720*.

Zhang, W. and Korf, R. E. (1996). A study of complexity transitions on the asymmetric traveling salesman problem. *Artificial Intelligence*, 81(1-2):223–239.