American Journal of Computer Science and Technology
2026; 09(3): 115-119
http://www.sciencepublishinggroup.com/j/ajcst
doi: 10.11648/j.ajcst.20260903.12

Methods for Counting and Enumerating Set Partitions

Arnav Khinvasara[1, 2], Alexander Pikovski[2, *]

[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, USA
[2]Research and Development, Unatech GmbH, Berlin, Germany

Email address:
ap@unatech.de (Alexander Pikovski)

To cite this article:
Arnav Khinvasara, Alexander Pikovski. (2026). Methods for Counting and Enumerating Set Partitions. *American Journal of Computer Science and Technology, 09*(3), 115-119. https://doi.org/10.11648/j.ajcst.20260903.12

Received: 30 July 2026; **Accepted:** 30 July 2026 **Published:** 23 September 2026

Abstract: Set partitions are arrangements of distinct objects into groups. After a brief review of the subject, we consider the task of counting and enumerating set partitions. The number of set partitions, known as Bell number, is a rapidly increasing number and does not have an explicit formula. We study approximate expressions for the Bell number given in the literature. We find that an asymptotic formula of Moser and Wyman gives a surprisingly accurate approximation to the Bell number even for small set sizes. Furthermore, a simple expression due to Berend and Tasssa can be conveniently used to approximate the Bell number for small set sizes. Next, we consider enumeration of set partitions. The problem of listing all set partitions arises in a variety of settings, in particular in combinatorial optimization tasks. Algorithms for enumerating all set partitions are reviewed. The focus is on non-recursive algorithms without Gray code constructions. We compare the classic algorithm of Hutchinson with three more modern ones. Empirically, it is found that all of them scale exponentially with the set size. While the exact compiler and optimization settings do matter, it can be concluded that the algorithm of Djokic et al. is the fastest one, thus it is recommended for practical use.

Keywords: Set Partitions, Bell Numbers, Algorithms, Benchmarking, Optimization

1. Introduction

A set partition is an arrangement of n distinct objects into groups. More formally, a partition of a set is a collection of non-overlapping sets (called groups or blocks) whose union is the original set. For example, the set $\{1, 2, 3\}$ has in total 5 partitions: $(\{1\}, \{2\}, \{3\})$, $(\{1, 2\}, \{3\})$, $(\{1, 3\}, \{2\})$, $(\{1\}, \{2, 3\})$, $(\{1, 2, 3\})$.

In practical problems, set partitions arise in a variety of settings. Usually one wants to find an optimal way to distribute n objects into groups according to certain criteria. If n is not too large, one can enumerate all possible set partitions and use a full search to find the optimal one. The total number of set partitions is a rapidly increasing number (see discussion in Sec. 3), and already for $n = 17$ we have about $8 \cdot 10^{10}$ partitions. Thus, for n larger than 17 or 18 (depending on the computational resources) listing all partitions is not feasible. Still, it is a problem of practical importance to list all set partitions for a given n, if it is not too large.

After briefly reviewing the application of set partitions, we discuss the number of set partitions (Bell numbers) and approximate formulas to calculate them. Then, we discuss and benchmark four algorithms for generating all set partitions, a classical one and three more modern ones. We conclude by recommending the algorithm with the best performance and give an outlook to topics outside the scope of this study.

The problem of enumerating set partitions in hardware, as opposed to software, is considered in [1].

A related problem is to arrange n distinct objects into groups size k. These are called restricted set partitions, in contrast to unrestricted set partitions which are the topic of the paper.

2. How Set Partitions Arise in Practice

A common problem which leads to set partitions is: What are the placements of n distinct items in at most n boxes? Similarly, set partitions arise in questions of the following type: What are the possible ways to serve n dishes on plates?

What are the possible ways to sit n people at tables? What are the possible "rhyme schemes" for n lines? (E. g. for 3 lines we have the 5 patterns abc, aab, aba, abb, aaa.)

Determining all set partitions has practical relevance in the setting of an optimization problem. Imagine you have n different items, which need to be packed and shipped. Each parcel may contain one or more items. There are certain constraints (size, weight, ...) which are case-specific. The way to go about solving this problem algorithmically is to check all possible ways to pack the items, calculating the case-specific constraints for each one, and then choose the optimal one.

Further applications of set partitions, among others to scheduling problems, are discussed in [2].

3. Number of Set Partitions (Bell Numbers)

The number of set partitions of a set of size n is called the Bell number B_n. The mathematical properties of Bell numbers are well-studied, see e. g. [3]. To calculate Bell numbers, one can use for example the following recursion (for further recursions, see [4]):

$$B_n = 1 + \sum_{k=1}^{n-1} \binom{n-1}{k} B_k. \tag{1}$$

This formula may be illustrated using the example of packing n items in boxes as follows. Look at one specific item. This item goes either alone in a box, or with k items. There are $\binom{n-1}{k}$ choices to find k neighbours for this item, and the remaining $n-k-1$ items can be packed in B_{n-k-1} ways in other boxes, for $k = 0 \ldots n-2$; the case $k = n-1$ (all items on one box) in addition contributes one. Thus in total, we have $\sum_{k=0}^{n-2} \binom{n-1}{k} B_{n-k-1} + 1$ ways to pack n items, which, after simplification, is the same as Eq. 1.

The exact value for B_n (see Table 1 for values up to $n = 20$) can obtained with computer-algebra packages. We used SymPy and Matlab for numerical calculations.

We now discuss several approximate expressions for the Bell numbers. A large variety of different asymptotic and approximate expressions are described in the literature [5–7]. We reviewed the different expressions and compared them numerically with exact values, to determine the most useful ones for practical use. Remarkably, we found that several asymptotic expressions for B_n give a very good approximation even at small values of n. The best one is the following.

The unique solution to the equation $\eta e^{\eta} = x$ (for positive x), is conventionally called Lambert's W function and denoted by $\eta = W(x)$. This function is not very common, but it is present in most numerical and symbolic libraries. The expression for Bell numbers obtained by Moser and Wyman [8] reads:

$$B_n \approx B_n^* = \frac{e^{nW(n)+n/W(n)-n-1}}{\sqrt{W(n)+1}} \times \left(1 - \frac{W(n)^2(2W(n)^2+7W(n)+10)}{24n(W(n)+1)^3}\right). \tag{2}$$

Table 1. *Comparing exact number of set partitions (Bell numbers) with the approximate expressions of Moser–Wyman (Eq. 2) and Berend–Tassa (Eq. 3).*

n	Set partitions B_n (Exact number)	Moser–Wyman Approximation	Berend–Tassa Approximation
1	1	1	1
2	2	2	2
3	5	5	5
4	15	15	15
5	52	52	52
6	203	203	212
7	877	877	957
8	4 140	4 143	4 781
9	21 147	21 161	26 107
10	115 975	116 040	154 507
11	678 570	$678.9 \cdot 10^3$	$984 \cdot 10^3$
12	4 213 597	$4.215 \cdot 10^6$	$6.70 \cdot 10^6$
13	27 644 437	$27.65 \cdot 10^6$	$48.5 \cdot 10^6$
14	190 899 322	$191.0 \cdot 10^6$	$372 \cdot 10^6$
15	1 382 958 545	$1.383 \cdot 10^9$	$3.01 \cdot 10^9$
16	10 480 142 147	$10.48 \cdot 10^9$	$25.6 \cdot 10^9$
17	82 864 869 804	$82.88 \cdot 10^9$	$229 \cdot 10^9$
18	682 076 806 159	$682.2 \cdot 10^9$	$2.14 \cdot 10^{12}$
19	5 832 742 205 057	$5.834 \cdot 10^{12}$	$20.8 \cdot 10^{12}$
20	51 724 158 235 372	$51.73 \cdot 10^{12}$	$211 \cdot 10^{12}$

While is an asymptotic expression for B_n, remarkably, it gives a very good approximation for all values of n. Specifically, comparing the result from Eq. 2 with the exact values numerically, we found that the relative difference $|B_n^* - B_n|/B_n$ is smaller than $3 \cdot 10^{-3}$ for all $2 \leq n \leq 50$. Moreover, for $n = 1 \ldots 7$, the integer part of B_n^* is exactly equal to B_n.

Berend and Tassa [9] proved the following upper bound on Bell numbers:

$$B_n < \overline{B}_n = \left(\frac{0.792\, n}{\log(n+1)} \right)^n , \tag{3}$$

valid for any n. Comparisons show that this bound is very tight for small n, and becomes less good for larger n. The formula for $\overline{B}_n$ is much simpler than Eq. 2 and can be used to estimate Bell numbers for not too large n.

The results are summarized in Table 1, where we show the exact value for B_n compared to the approximations of Moser–Wyman and Berend–Tassa. The conclusion is that for smaller n, the simple expression of Eq. 3 gives an excellent approximation while for larger n one should use Eq. 2.

4. Algorithms for Listing Set Partitions

Finding the optimal arrangement of objects requires, as discussed above, in some cases a listing of all partitions of a set. The problem of finding an efficient algorithm for enumerating set partitions was repeatedly treated in the literature. However, a comparison of the different proposals, with an eye towards practical use, is lacking.

The recursion of Eq. 1 and similar ones suggest that the problem of generation of all set partitions can be solved recursively. While this is true, recursive algorithms in practice are less efficient than non-recursive (iterative) ones. The main reason is that repeated function calling creates an overhead which is also very much system- and compiler-specific. It is clear that, generally, iterative algorithms outperform recursive ones. Thus we only benchmark the non-recursive algorithms.

A class of algorithms for generating set partitions are those based on Gray codes. Here, a Gray code is a one-to-one mapping from $\{1, \ldots, B_n\}$ to the set partitions for set size n, having certain properties. In other words, we get an indexing scheme (of a special type) for set partitions. Gray-code base approaches often deal with both restricted and unrestricted set partitions [10]. However, we do not consider Gray code constructions here.

The oldest algorithm for calculating set partitions is due to Hutchinson [11]. This classical algorithm is also presented in the recent volume of the influential book by Knuth [5]. However, Hutchinson's algorithm is definitely not the most efficient one, and we do not recommend its use in practical calculations (see discussion in Sec. 5).

A fast algorithm for generating all set partitions was presented by Semba [12]. Another fast algorithm was devised by Er [13] (incidentally, using ideas from Gray codes but without explicitly constructing them). Later Djokic et. al. presented another fast algorithm for generating set partitions [14].

A pedagogical exposition of the mentioned algorithms is given in [15].

5. Benchmarking of Algorithms

Since there is no clear theoretical analysis of the running time of algorithms for set partitions, we performed an empirical benchmarking. (The empirical studies presented in [13, 14] are clearly out of date.)

We have tested the algorithms [11–14]. The algorithms were tested both on Linux and on Windows operating systems. The code was written in C, to ensure lightweight binaries and fast running times. The compilers used were GNU C Compiler (GCC), Intel C Compiler (ICC) and Microsoft Visual C (MSVC).

Comparing different compilers on the same machine, we get the following pattern. For similar optimization levels, Hutchinson's algorithm runs significantly faster on GCC compiler than using the ICC compiler. For Semba's and Er's algorithm, GCC is faster than ICC, while for Djokic's the GCC is much slower than ICC.

Comparing different operating systems (and compilers) on the same machine, we find that Linux-based code (GCC, ICC) is faster than Windows-based (MSVC); the former can be up to a factor of 2 faster.

The compiler optimization level plays a decisive role, however, the difference between optimization level 2 and 3 is not large. Turning on some additional hardware-specific optimizations does not always lead to faster code.

Now, comparing different algorithms with the same compiler (ICC), with highest compiler optimization. Results for running times are shown in Figure 1. Hutchinson's algorithm is significantly slower than the other three. The algorithms of Semba and Er are faster, and that of Djokic et al. is the fastest one. It is clearly seen that the running time for all four algorithms scales exponentially with n.

For practical purposes, we recommend the algorithm of [14]. The algorithm is relatively short and easy to implement. Compiler optimization should be set at least at level 2. On Linux, the Intel Compiler (ICC) for this algorithm gives faster code than GNU C Compiler (GCC).

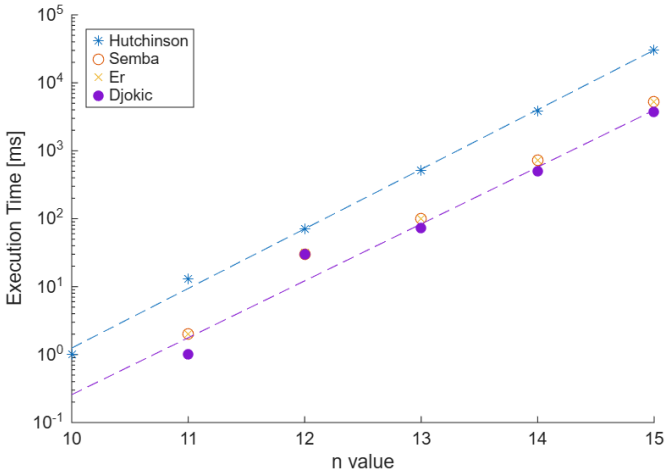

Figure 1. *Execution time for the studied algorithms, for different n. Time is shown in logarithmic scale. Dashed lines are linear fits to the data points. ICC compiler with highest optimization level was used on Intel Core i7-11700 @ 2.50 GHz CPU.*

6. Conclusions and Outlook

After an introduction to set partitions and some general background, we have presented useful approximation formulas for the calculation of the number of set partitions (Bell numbers), both for small and large set sizes. Three state-of-the-art nonrecursive algorithms [12–14] for generating set partitions were benchmarked, with a comparison with Hutchinson's classical algorithm [11]. As a result, the algorithm of Djokic et al. [14] is recommended for practical use.

A problem closely related to finding all set partitions is that of finding all partitions with a maximal block size p, known as restricted set partitions. Some of the algorithms for unrestricted set partitions can be modified to yield restricted set partitions. Together with a review and benchmarking of Gray-code based algorithms, this warrants a separate investigation. Furthermore, it would be useful to study parallelizable algorithms for listing set partitions.

ORCID

0000-0003-3452-194X (Alexander Pikovski)

Abbreviations

B_n	n-th Bell Number (Number of Set Partitions)
W	Lambert's W Function
GCC	GNU C Compiler
ICC	Intel C Compiler
MSVC	Microsoft Visual C Compiler

Author Contributions

Arnav Khinvasara: Formal Analysis, Investigation, Software

Alexander Pikovski: Conceptualization, Investigation, Methodology, Writing – original draft, Writing – review & editing

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] J. T. Butler and T. Sasao, "High-speed hardware partition generation," ACM Trans. Reconfigurable Technol. Syst., vol. 7, Dec. 2014. https://doi.org/10.1145/2629472

[2] R. K. Hankin and L. J. West, "Set partitions in R," Journal of Statistical Software, vol. 23, pp. 1–12, 2008. https://doi.org/10.18637/jss.v023.c02

[3] G.-C. Rota, "The number of partitions of a set," The American Mathematical Monthly, vol. 71, no. 5, pp. 498–504, 1964. https://doi.org/10.1080/00029890.1964.11992270

[4] M. Z. Spivey, "A generalized recurrence for Bell numbers," J. Integer Seq., vol. 11, no. 08.2, p. 5, 2008.

[5] D. E. Knuth, Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Boston: Addison-Wesley, 2011.

[6] T. Mansour, Combinatorics of set partitions. Boca Raton: CRC Press, 2013. https://doi.org/10.1201/b12691

[7] J. Grunwald and G. Serafin, "Explicit bounds for Bell numbers and their ratios," Journal of Mathematical Analysis and Applications, vol. 549, no. 2, p. 129527, 2025. https://doi.org/10.1016/j.jmaa.2025.129527

[8] L. Moser and M.Wyman, "An asymptotic formula for the Bell numbers," Trans. Roy. Soc. Can., vol. 49, no. Series III, Sec. III, pp. 49–54, 1955.

[9] D. Berend and T. Tassa, "Improved bounds on Bell numbers and on moments of sums of random variables," Probability and Mathematical Statistics, vol. 30, no. 2, pp. 185–205, 2010.

[10] S. Kawano and S. Nakano, "Constant time generation of set partitions," IEICE Transactions on Fundamentals, vol. E88-A, pp. 930–934, April 2005. https://doi.org/10.1093/ietfec/e88-a.4.930

[11] G. Hutchinson, "Partioning algorithms for finite sets," Communications of the ACM, vol. 6, no. 10, pp. 613–614, 1963. https://doi.org/10.1145/367651.367661

[12] I. Semba, "An efficient algorithm for generating all partitions of the set {1, 2,..., n}," Journal of Information Processing, vol. 7, no. 1, pp. 41–42, 1984.

[13] M. Er, "A fast algorithm for generating set partitions," The Computer Journal, vol. 31, no. 3, pp. 283–284, 1988. https://doi.org/10.1093/comjnl/31.3.283

[14] B. Djokić, M. Miyakawa, S. Sekiguchi, I. Semba, and I. Stojmenović, "A fast iterative algorithm for generating set partitions," The Computer Journal, vol. 32, no. 3, pp. 281-82, 1989. https://doi.org/10.1093/comjnl/32.3.281

[15] G. Stamatelatos and P. S. Efraimidis, "Lexicographic enumeration of set partitions." arXiv preprint 2105.07472, https://doi.org/10.48550/arXiv.2105.07472, 2021