USING GPUS AND LLMS CAN BE SATISFYING FOR NONLINEAR REAL ARITHMETIC PROBLEMS

Christopher Brix, Julia Walczak, Nils Lommen, Thomas Noll
RWTH Aachen University
{brix,lommen,noll}@cs.rwth-aachen.de, julia.walczak@rwth-aachen.de

ABSTRACT

Solving quantifier-free non-linear real arithmetic (NRA) problems is a computationally hard task. To tackle this problem, prior work proposed a promising approach based on gradient descent. In this work, we extend their ideas and combine *LLMs* and *GPU acceleration* to obtain an efficient technique. We have implemented our findings in the novel SMT solver *GANRA* (**G**PU **A**ccelerated solving of **N**onlinear **R**eal **A**rithmetic problems).[1] We evaluate GANRA on two different NRA benchmarks and demonstrate significant improvements over the previous state of the art. In particular, on the Sturm-MBO benchmark, we can prove satisfiability for more than five times as many instances in less than 1/20th of the previous state-of-the-art runtime.

1 INTRODUCTION

Satisfiability modulo theories (SMT) is an important framework to formalize many mathematical problems. An SMT problem combines a formal theory with traditional satisfiability (SAT) solving. To be precise, an SMT problem is constructed by replacing the Boolean variables of the SAT problem with formulas over the given formal theory. Nowadays, many problems in various areas of computer science, such as software verification, automated theorem proving, or analysis of hybrid systems, are reduced to SMT problems. In this work, we focus on quantifier-free non-linear real arithmetic (NRA) as the underlying formal theory. NRA considers polynomial equations and inequations over the domain $\mathbb{R}$ of real numbers. For example, consider the following simple SMT problem:

$$\phi(x_1, x_2, y_1, y_2) = \left(x_1^2 + x_2^2 = 1 \wedge y_1^2 + y_2^2 = 1 \wedge (x_1 - y_1)^2 + (x_2 - y_2)^2 \geq 1\right) \tag{1}$$

This SMT problem ϕ encodes two points (x_1, x_2) and (y_1, y_2) in the 2-dimensional plane on the unit sphere such that their distance is at least 1. This problem is also known as the "Kissing number problem". Here, for example, $(x_1, x_2) = (1, 0) \in \mathbb{R}^2$ and $(y_1, y_2) = (0, -1) \in \mathbb{R}^2$ is a model that satisfies ϕ, i.e., $\phi(1, 0, 0, -1) = \text{TRUE}$.

There already exist numerous techniques to prove or disprove satisfiability of such NRA problems, e.g., most prominently, cylindrical algebraic decomposition (CAD) Collins (1975), virtual substitution Weispfenning (1988; 1997), or approaches employing Gröbner bases Stengle (1974). However, complete techniques such as CAD have doubly-exponential worst-case runtime. Thus, in Cimatti et al. (2022); Lipparini (2024); Lipparini & Ratschan (2025); Liu et al. (2023), the authors proposed an approach based on gradient descent to automatically prove satisfiability of NRA problems (and also problems with transcendental functions) efficiently. More precisely, a function $f : \mathbb{R}^m \to \mathbb{R}$ is constructed of the NRA problem ϕ (where m is the number of variables in ϕ) such that if $\boldsymbol{x}$ is a model for the original NRA problem, then $f(\boldsymbol{x}) \leq 0$. To find the roots of f, classical gradient descent is applied. In this work, we extend their ideas and utilize GPUs to increase the performance. To this end, we "group" similar operations such that GPU acceleration is applicable. For example, all multiplications x_1^2, x_2^2, etc. could be performed in parallel by GPUs. In order to make this "grouping" automatable, we query LLMs. To be precise, we use the LLM OpenAI o1-preview. So, we combine LLMs and GPUs, and make use of their recent progress to solve NRA problems. Our approach is implemented as a prototype in the novel tool GANRA (**G**PU **A**ccelerated solving of **N**onlinear **R**eal **A**rithmetic).[1] We evaluated our tool on two sets of benchmarks: The Kissing

[1] `https://github.com/ChristopherBrix/GANRA`

benchmark set (Jovanović & de Moura, 2013) and the Sturm-MBO benchmark set (Sturm & Weber, 2008). Here, our prototype implementation already showed promising results. To discuss the strengths of our approach further, we introduced a customizable benchmark set based on the Sturm-MBO benchmarks.

Contributions: In the following, we summarize our main contributions:

- We establish "grouping" of similar operations as a key element to benefit from GPU acceleration. This is demonstrated in an empirical evaluation on two sets of benchmarks.
- We demonstrate that LLMs are capable of identifying patterns in benchmarks that allow for such operation grouping. This removes the need to manually encode similar operations, and motivates further research into optimizing this automatic detection.
- We provide a customizable benchmark set that allows for a detailed analysis of the performance of NRA-tools on increasingly complex polynomials.
- Our techniques are implemented in the novel tool GANRA. GANRA is – to the best of our knowledge – the first SMT solver that uses a combination of formal methods and LLMs to solve NRA problems profiting of GPU acceleration.

Structure: First, we present related work in Section 2. In Section 3, we briefly recap how gradient descent can be used to prove satisfiability of NRA problems. Afterwards, we introduce manual optimization techniques to improve GPU acceleration in Section 4. In Section 5, we present our automatic approach GANRA in which we query the LLM OpenAI o1-preview to perform the optimization for the GPU. We demonstrate the strengths of our approach in an empirical evaluation on the Kissing and Sturm-MBO benchmarks in Section 6. Finally, we discuss future work in Section 7 and conclude in Section 8.

2 Related Work

Most state-of-the-art SMT solvers implemented different quantifier elimination techniques to solve NRA formulas. A survey on common quantifier elimination techniques such as cylindrical algebraic decomposition (CAD) Collins (1975), virtual substitution (Weispfenning, 1988; 1997), and quantifier elimination using Gröbner bases (Stengle, 1974) is given by Dolzmann et al. (1999). Complete quantifier elimination techniques like CAD are the centerpiece of many SMT solvers (see, e.g., CVC5 (Barbosa et al., 2022), SMT-RAT (Corzilius et al., 2015), Yices2 (Dutertre, 2014), or Z3 (de Moura & Bjørner, 2008)). However, such complete techniques have a doubly-exponential worst-case runtime.

Hence, we use a completely different (incomplete but efficient) approach in this paper. We have implemented a technique based on gradient descent in our tool GANRA. In contrast to complete techniques, gradient descent can only be used to prove (but not to disprove) NRA formulas. The idea of identifying satisfying assignments of NRA problems by employing gradient descent was first proposed by Cimatti et al. (2022); Lipparini (2024) and independently by Liu et al. (2023). In both cases, it was demonstrated that "gradient descent SMT solving" is a strong alternative to the existing techniques. We are now continuing this line and adapt this technique such that it benefits from GPU acceleration. Similar techniques are already showing promising results for bounded model checking (Osama & Wijs, 2021) or SAT solving (Osama et al., 2024).

There exist already different approaches to parallelize SMT solvers. However, to the best of our knowledge, there does not exist an SMT solver optimized for GPUs. There are often two types of parallelization: Portfolio and divide-and-conquer solvers. Portfolio solvers apply various techniques in parallel, e.g., SMT-RAT (Corzilius et al., 2015), SMTS (the parallel version of OpenSMT) (Marescotti et al., 2018), or Z3 (Wintersteiger et al., 2009) use a portfolio of techniques. They often improve efficiency by sharing intermediate results of different threads. In contrast, divide-and-conquer approaches partition the SMT formula into smaller subproblems, e.g., (Zhao et al., 2024; Wilson et al., 2023) in CVC5, or (Hyvärinen et al., 2015) on top of OpenSMT.

3 GRADIENT DESCENT AS A GPU-ACCELERATED HEURISTIC

Common NRA verification tools such as Z3 or CVC5 do not support GPUs to accelerate the computation. Cimatti et al. (2022); Liu et al. (2023) proposed to encode a given NRA formula as a function over the reals, such that every satisfying assignment to the original NRA formula also constitutes a root of the corresponding function. This allows to perform a gradient descent along the generated function. For each identified root, the corresponding input can be tested to determine whether it is a valid satisfying assignment.

3.1 LOGIC-TO-OPTIMIZATION TECHNIQUE

To utilize gradient descent to find potential satisfying inputs of the NRA problem, we must convert the original formula $\phi(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^m$ into a function $f : \mathbb{R}^m \to \mathbb{R}$. This conversion is designed to guarantee that any model $\boldsymbol{x}$ of ϕ (any $\boldsymbol{x}$ that satisfies ϕ) has $f(\boldsymbol{x}) \leq 0$. Notably, the inverse does not need to hold: For some $\boldsymbol{x}'$ with $f(\boldsymbol{x}') \leq 0$, $\boldsymbol{x}'$ may or may not be a model of ϕ. This concept of converting the formula into a corresponding function was introduced by Cimatti et al. (2022); Liu et al. (2023). Note that our L2O (logic-to-optimization) transformations (2) to (4) differ from (Cimatti et al., 2022), and all definitions differ from (Liu et al., 2023). In particular, we introduce a parameter ϵ that increases the set of inputs $\boldsymbol{x}$ that yield $\text{L2O}_\epsilon(\phi(\boldsymbol{x})) \leq 0$. We provide a comparison with both proposals, as well as an evaluation of the impact of the choice of ϵ, in Section 6.4, highlighting the importance of choosing an L2O transformation that facilitates the gradient descent. As in Cimatti et al. (2022), we assume that the formula is given as nested conjunctions and disjunctions, as well as constraints over real functions of the form $p(\boldsymbol{x}) \sim 0$ for $\sim \in \{\leq, <, =\}$. We enforce this by rewriting the formulas accordingly. For example, we remove all negations by flipping the relevant relation comparisons.

$$\text{L2O}_\epsilon(p(\boldsymbol{x}) = 0) \stackrel{\text{def}}{=} \max\{|p(\boldsymbol{x})| - \epsilon, 0\} \tag{2}$$

$$\text{L2O}_\epsilon(p(\boldsymbol{x}) < 0) \stackrel{\text{def}}{=} \max\{p(\boldsymbol{x}), -\epsilon\} \tag{3}$$

$$\text{L2O}_\epsilon(p(\boldsymbol{x}) \leq 0) \stackrel{\text{def}}{=} \max\{p(\boldsymbol{x}), 0\} \tag{4}$$

$$\text{L2O}_\epsilon(\phi_1(\boldsymbol{x}) \wedge \phi_2(\boldsymbol{x})) \stackrel{\text{def}}{=} \text{L2O}_\epsilon(\phi_1(\boldsymbol{x})) + \text{L2O}_\epsilon(\phi_2(\boldsymbol{x})) \tag{5}$$

$$\text{L2O}_\epsilon(\phi_1(\boldsymbol{x}) \vee \phi_2(\boldsymbol{x})) \stackrel{\text{def}}{=} \text{L2O}_\epsilon(\phi_1(\boldsymbol{x})) \cdot \text{L2O}_\epsilon(\phi_2(\boldsymbol{x})) \tag{6}$$

Note that none of the two benchmarks we evaluated depend on Definition 6, we provide it solely for completeness.

Example For our running example in Equation 1, the L2O conversion yields the following result:

$$\begin{aligned}
&\text{L2O}_\epsilon(\phi(x_1, x_2, y_1, y_2)) \\
&= \text{L2O}_\epsilon(x_1^2 + x_2^2 = 1) + \text{L2O}_\epsilon(y_1^2 + y_2^2 = 1) + \text{L2O}_\epsilon((x_1 - y_1)^2 + (x_2 - y_2)^2 \geq 1) \\
&= \text{L2O}_\epsilon(x_1^2 + x_2^2 - 1 = 0) + \text{L2O}_\epsilon(y_1^2 + y_2^2 - 1 = 0) \\
&\qquad + \text{L2O}_\epsilon(1 - ((x_1 - y_1)^2 + (x_2 - y_2)^2) \leq 0) \\
&= \max\{|x_1^2 + x_2^2 - 1| - \epsilon, 0\} + \max\{|y_1^2 + y_2^2 - 1| - \epsilon, 0\} \\
&\qquad + \max\{1 - ((x_1 - y_1)^2 + (x_2 - y_2)^2), 0\}
\end{aligned}$$

3.2 ENSURING SOUNDNESS

As described above, the Logic-to-Optimization approach ensures that every satisfying assignment of ϕ guarantees $\text{L2O}_\epsilon(\phi(\boldsymbol{x})) \leq 0$, but not every assignment $\boldsymbol{x}'$ with $\text{L2O}_\epsilon(\phi(\boldsymbol{x}')) \leq 0$ necessarily satisfies $\phi(\boldsymbol{x}')$. Therefore, after identifying a potential candidate for a satisfying assignment, we verify its correctness by running Z3 on a modified version of the problem instance, where each input variable is bounded by a small region around the identified candidate assignment. If Z3 rejects the assignment or times out, we restart the gradient descent from new random points.

If the benchmark does not contain any equality constraints, we can simplify this process: Assume ϕ consists of a single constraint $p \sim 0$ with $\sim \in \{<, \leq\}$. Then, we can test if $\text{L2O}_\epsilon(\phi(\boldsymbol{x})) \sim 0$ holds for our found candidate solution $\boldsymbol{x}$. Here, this is a sufficient condition: Any such input $\boldsymbol{x}$ is a valid model of ϕ. For conjunctions and disjunctions over multiple such constraints, we can perform this test individually and compute the respective conjunction and disjunction. However, this is not a sufficient condition for constraints $p = 0$. Thus, we use the SMT solver Z3 to prove correctness of our candidate $\boldsymbol{x}$ for equalities.

4 Manual Optimization for GPU Acceleration

GPUs can accelerate computations by performing a large number of similar operations in parallel, in particular matrix-matrix multiplications. Therefore, to maximize the speedup gained from using a GPU, it is crucial to structure the required operations in such a way that similar operations are grouped.

We identified two possible axes for this optimization:

1. Batching. Instead of sampling a single initial assignment $\boldsymbol{x}$ and optimizing it via gradient descent, multiple individual initial assignments can be sampled. Propagating them through $\text{L2O}_\epsilon(\phi(\boldsymbol{x}))$ can be done in parallel, effectively scaling the computation speed by the chosen batch size.
2. Grouping. If the computation of the L2O function requires to perform the same operation on different terms, these can be grouped, similar to batching. Crucially, this does not increase the number of samples evaluated, but reduces the required processing time per batch. E.g., for the function $f(\boldsymbol{x}) = (x_1 + x_2) \cdot (x_3 + x_4)$, the two sums $x_1 + x_2$ and $x_3 + x_4$ can be computed in parallel. This reduces the required computation to two sequential operations, instead of three.

In the following sections, we detail the grouping optimizations we performed for the Kissing and Sturm-MBO benchmarks. An evaluation of the importance of these optimizations is given in Section 6, where we compare the parallelized version to a sequential computation.

4.1 Kissing

The Kissing benchmark (Jovanović & de Moura, 2013) consists of 45 instances. Each instance encodes a kissing number problem: One searches for the position of N points on an M-dimensional unit sphere such that the pairwise distance between these points is greater than or equal to 1. For N unit spheres with $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^M$, this is encoded by

$$\phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \left(\bigwedge_{n \in \{1,\ldots,N\}} \boldsymbol{x}_n^T \boldsymbol{x}_n = 1 \wedge \bigwedge_{\substack{m,n \in \{1,\ldots,N\} \\ m<n}} (\boldsymbol{x}_n - \boldsymbol{x}_m)^T (\boldsymbol{x}_n - \boldsymbol{x}_m) \geq 1 \right)$$

From this vectorized formulation, it is immediately obvious that the inner subtractions and all multiplications can be performed in parallel.

4.2 Sturm-MBO

The Sturm-MBO benchmark (Sturm & Weber, 2008) consists of 405 instances, of which we select the 105 and 15 instances respectively known or unknown to be satisfiable[2]. Each instance defines a subset of the input variables $\{h_1, \ldots, h_6, j_2\}$ restricted to the positive reals. A satisfying assignment is a root of a multilinear polynomial. Here, each monomial of the polynomial is a scalar integer value, multiplied by N sampled instances of the variables $\{h_1, \ldots, h_6\}$. As N is often greater than 6, the constructed product often consists of powers of h_i. Finally, each product is optionally multiplied by j_2 with a power of up to N. E.g., for $N = 10$, the polynomial function has therefore the form

$$f(\boldsymbol{h}) = 2 \cdot h_1^4 \cdot h_2^2 \cdot h_3 \cdot h_5^3 + 13 \cdot h_1 \cdot h_2^2 \cdot h_3 \cdot h_4^3 \cdot h_5^2 \cdot h_6 \cdot j_2^8 + \ldots \tag{7}$$

[2] As indicated in their status tag

We optimize the computation for this benchmark by first precomputing all values $\bigcup_{i \in \{1,\ldots,6\}} \{h_i^n \mid n \in \{1,\ldots,N\}\} \cup \{j_2^n \mid n \in \{1,\ldots,N\}\}$. As these are independent of each other, they can be computed in parallel. Next, we initialize a vector with the scalar integers in each product. Then, for $i \in \{1,\ldots,6\}$, we select the required power of h_i in each product and copy it into a new tensor. We do the same for the required powers of j_2. This can be done in parallel for all products. Subsequently, these eight tensors (one for scalars, six for h_i, and one for j_2) are multiplied with each other, resulting in one tensor with the values of each product. Finally, the tensor is summed up, to get the required sum of all products.

A satisfying assignment of the benchmark requires the sum of products to be exactly equal to zero. Since finding those inputs via gradient descent may often be infeasible, as the optimization may constantly overshoot the actual assignment, we transform the benchmark further: We duplicate every input variable x_i to $\underline{x_i}$ and $\overline{x_i}$. We define the polynomials for both sets of variables and add the constraints $\underline{x_i} \leq 0$ and $\overline{x_i} \geq 0$ respectively. If this new benchmark version can be satisfied, the intermediate value theorem guarantees that the original benchmark could also be satisfied. One could then use interval iteration to find precise lower and upper bounds to the original x_i variables, potentially enabling other solvers to find the precise values using the limited search space.

5 LLM-Based Optimization for GPU Acceleration

The optimizations described in Sections 4.1 and 4.2 were performed by manually inspecting the benchmark to identify the underlying patterns. This is a time consuming approach that can only be justified if there are only a few different benchmark categories to evaluate, and if each category consists of many similarly structured problem instances. As the number of benchmarks increases, a manual optimization may become infeasible.

Therefore, we investigated whether the manual optimization of the computation process could be offloaded to an LLM. Ideally, the LLM would identify the underlying pattern of each benchmark, and provide the corresponding Python code to compute the L2O formula in an optimized manner. If this succeeds, the GPU acceleration becomes feasible even for large sets of diverse benchmark categories.

The usage of LLMs to generate the Python implementation of the L2O formula has two main risks: Syntactic errors and semantic errors. Crucially, neither of them can cause GANRA to produce unsound outputs, as they would in the worst case lead to non-termination:

1. Syntactic errors: The LLM may fail to generate Python code that can be executed. This can trivially be detected by trying to execute it - should the process crash, the LLM may either be queried to generate an updated code block, or grouping of computations may be disabled. Then, one could either still use GPU acceleration based on the non-grouped implementation of the L2O formula, that performs all operations in the order they are trivially specified. Alternatively, one may fall back to a solver such as Z3 that does not rely on gradient descent.

2. Semantic errors: The LLM may generate code that is syntactically correct, but is not equivalent to the L2O formula. This kind of error may be detected by comparing the results of the LLM-generated code on a few inputs against the slow but known good ground-truth implementation without grouping. Crucially, even if this test fails to identify a semantic error, the overall process is still sound: Computing the roots of the incorrect formula will yield candidates that are not actually satisfying assignments to the original formula. However, as these are always checked for correctness, they would always be discarded as a spurious assignment. Therefore, our solver would never return an actual satisfying assignment, and eventually time out.

The LLM is prompted by providing two example instances and asking it to generate efficient pytorch code to compute the required functions. We provide the exact LLM prompt in Appendix A.3. In the current version of GANRA, the reply of the LLM is manually inspected for the code block, which is then copied into the source code of GANRA without further edits. This process could be fully automated by employing an API to the LLM and automatically detecting and executing code blocks.

In our experiments, the LLM is presented with two example inputs from the benchmark category and is asked to identify patterns and extrapolate them to write efficient code. The prompt may be improved by providing additional information such as statistics on term reuse. For example, using Python's SymPy package (Meurer et al., 2017), the number of occurrences of each term may be computed and passed as additional input. Exploring such extensions is left for future work.

5.1 Case Studies

We provide the LLM generated code for the two benchmarks in Appendix A.3. Here, we want to highlight some of the correctly identified grouping opportunities, as well as missed opportunities that cause the LLM generated code to be suboptimal.

Kissing The LLM finds some of the possible groupings, but not all of them: It computes the multiplication of $\boldsymbol{x}_n^T \boldsymbol{x}_n$ for all $n \in \{1, \ldots, N\}$ in parallel. Then, it computes all pairwise squared distances between all $\boldsymbol{x}_n, \boldsymbol{x}_m$ vectors. Crucially, it does not take into account that only the distances for $m < n$ are required, doubling the necessary computations. To account for that, only the relevant subset of the pairwise distances is then selected for the final result.

Sturm-MBO The initial generated code contained a syntax error. We replied to the LLM by copying this error verbatim, which caused it to provide an updated version of the code. Such a feedback loop could be done automatically.

The LLM successfully identifies the potential to speed up the computation by grouping the computation of each product, instead of performing the multiplications sequentially. However, if multiple products contain the same power of the same variable (e.g., x_2^2 in Equation 7), the LLM does not precompute this value. Instead, it computes the power for each product from scatch, significantly increasing the computation overhead.

6 Results

We compare the performance of GANRA against Z3, CVC5, ugotNL (Lipparini & Ratschan, 2025)[3] and NRAgo (Liu et al., 2023) on two benchmarks from the SMT-LIB benchmark set (Barrett et al., 2016): Kissing (Jovanović & de Moura, 2013) and 20161105-Sturm-MBO (Sturm-MBO) (Sturm & Weber, 2008). For each benchmark, we evaluate the following setups for GANRA:

1. an L2O process with $\epsilon = 10^{-4}$ and sequential, non-optimized code,
2. an L2O process with $\epsilon = 0$ and manually optimized code,
3. an L2O process with $\epsilon = 10^{-4}$ and manually optimized code, and
4. an L2O process with $\epsilon = 10^{-4}$ and LLM-generated code to compute real functions.

All experiments were performed using a single CPU core with 5G RAM and an NVIDIA H100 GPU. Unless otherwise noted, the timeout for each instance was 10 minutes. We always randomly sample a batch of 10 000 initial assignments from the interval $[-1, 1]$ and optimize them in parallel, using batching. The gradient descent was done using the Adam optimizer introduced in (Kingma & Ba, 2017) and a learning rate of 10^{-3}. After each update, values were projected back to $[-1, 1]$.

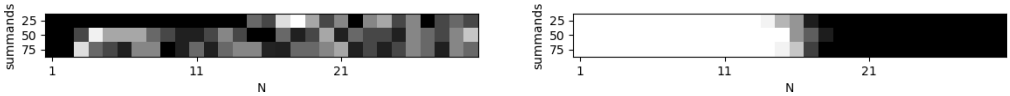
6.1 Kissing

As comparing the average runtime of tools that successfully process a different set of inputs is unfair, we report the average runtime over those instances that can be solved by all tools that solve at least 30 instances. Table 1 summarizes the results. We outperform all other tools both in terms of total number of found satisfiable instances, and the average runtime to find them. Interestingly, the LLM-based implementation performs on par with the manually optimized one, even though it includes unnecessary computations (see Section 5.1). This indicates that the LLM does not necessarily need to find the optimal solution, as even partially optimized code can already exhibit great performance.

[3] Note that this is an updated version of the solver presented in (Cimatti et al., 2022) that achieves stronger results than the initial version.

		Z3	CVC5	UGOTNL	NRAgo	GANRA (ours)			
						seq.	manual		LLM
					$\epsilon =$	10^{-4}	0	10^{-4}	10^{-4}
Kis.	SAT [↑]	34	16	27	39	39	18	**40**	**40**
	avg. runtime [s] [↓]	38.01	-	-	24.72	30.45	-	**9.13**	9.61
MBO	SAT [↑]	1	0	10	6	10	**57**	**57**	38
	avg. runtime [s] [↓]	-	-	139.08	-	333.62	**6.39**	14.86	35.92

Table 1: Toolkit evaluation on the Kissing and Sturm-MBO benchmarks. We report the average runtime of those instances that could be verified by all tools that can prove satisfiability for at least 30 (Kissing) and 10 (Sturm-MBO) instances. ↑ (↓): Higher (lower) values are better.



(a) Z3 (b) GANRA

Figure 1: Comparison of Z3 and GANRA on Sturm-MBO-like problems with varying number of summands and N. White indicates 0% success at finding a satisfying input, black 100%.

Using $\epsilon = 0$ causes the performance of GANRA to deteriorate. We hypothesize that this is because the Kissing benchmark includes multiple equality constraints. Without an $\epsilon > 0$ in Equation 2, the L2O formula would only have pointwise roots, making the optimization problem very hard, as it would require to find the precise corresponding scalar inputs. By setting $\epsilon > 0$, we increase the area in which the gradient descent may be stopped, to allow testing the potential satisfiable input.

6.2 STURM-MBO

Table 1 summarizes the results for the Sturm-MBO benchmark. Here, all optimized versions of GANRA outperform all baselines. The performance of the LLM-based version lacks behind the manually optimized one, but can still prove satisfiability for almost four times as many instances as UGOTNL, the best existing tool.

Note that for GANRA, we use a modified version of the Sturm-MBO benchmark (see Section 4.2): Instead of finding a root of the given polynomial function, we find inputs that yield positive/negative outputs. This simplifies the optimization process, while concrete inputs could be found using interval iteration. The other tools perform slightly worse on this modified benchmark version, so here, we report their performance on the original instances. We provide a comparison of the performances across both versions in Appendix A.2.

6.3 ABLATION STUDY ON STURM-MBO LIKE PROBLEMS

In order to better demonstrate the domain in which GANRA outperforms conventional approaches, we create a new Sturm-MBO like benchmark, with tunable hyperparameters. We generate new instances (according to the pattern discussed in Section 4.2) for P products in the sum, and N being the sum of all exponents of $h_1, \dots, h_6$ per product. We explore all combinations of $P \in \{25, 50, 75\}$ and $N \in \{1, \dots 30\}$. We provide additional information in Appendix A.1. As we do not know apriori whether the generated instances will be satisfiable or unsatisfiable, we attempt to find a satisfying input using both Z3 and GANRA with a shortened timeout of three minutes. Any instance that cannot be shown to be satisfiable by at least one of the two solvers is discarded. We repeat this process until we have 10 known satisfiable instances per hyperparameter combination.[4]

In Figure 1 we report the percentage of verified instances per tool. It is evident that GANRA shines particularly for a higher value of N, i.e., for complicated products. This is intuitive, as more complex

[4]If we cannot generate 10 instances within one hour, we terminate and use a reduced set of instances for this hyperparameter combination.

	Kissing			Sturm-MBO		
L2O version	GANRA	NRAgo	UGOTNL	GANRA	NRAgo	UGOTNL
SAT [↑]	**40**	38	39	**57**	**57**	56
avg. runtime [s] [↓]	9.62	**6.79**	12.69	**38.42**	42.92	113.01

Table 2: Comparison of the different L2O definitions proposed in our work vs. (Cimatti et al., 2022) and (Liu et al., 2023). We integrated their L2O definitions into GANRA, so all results benefit from GPU acceleration. ↑ (↓): Higher (lower) values are better.

polynoms complicate the solving process for Z3, but gradient descent can be applied regardless. Future work may investigate heuristics that could be used to predict whether a given benchmark would benefit from a gradient descent-based approach.

6.4 Alternative L2O Transformations

To ablate the impact of the given L2O definition, we integrate the L2O definitions from (Cimatti et al., 2022) and (Liu et al., 2023) in GANRA. In Table 2, we see the respective performances of all three definitions on both benchmarks. On the Kissing benchmark, all three versions perform with different tradeoffs between number of found satisfiable inputs and average runtime. On the Sturm-MBO benchmark however, the UGOTNL L2O definition is significantly slower than both our proposed definition and the one by (Liu et al., 2023) in NRAgo. We propose to investigate these differences in more detail in future work, to potentially find a new L2O definition that merges the individual benefits of each existing version.

7 Future Work

Even though we successfully demonstrate that LLMs are capable of writing efficient code for the computation of given formulas, they also lack behind the performance of hand-written code. This demonstrates two orthogonal research directions: On the one hand, a tuning of the LLM prompt, or a better feedback loop that allows the LLM to iteratively improve its code may be able to increase the performance of the LLM-based implementation. On the other hand, a more thorough analysis of a larger set of benchmarks may unearth underlying patterns that would allow to write hand-optimized code that directly supports a wide variety of different benchmarks. This would potentially remove the need for LLMs entirely.

We also note that our implementation does not benefit from any of the improvements implemented in UGOTNL and NRAgo, such as Basinhopping (Wales & Doye, 1997) to improve the process of finding roots, or the topological degree test (Cho & Chen, 2006; Franek & Ratschan, 2012) to remove our reliance on Z3 for verification of found candidates. Furthermore, GANRA has no ability to prove unsatisfiability. For unsat instances, it would always result in a timeout, making them indistinguishable from hard but satisfiable instances. Ultimately, the concept of GANRA should therefore be integrated with a complete solver by calling GANRA as part of a portfolio approach.

Finally, we note that our evaluation focused on two benchmarks only. Evaluating GANRA on more benchmarks, preferably the entire set of available quantifier-free NRA problems, may provide valuable insights into its strength and failure cases.

8 Conclusion

We have evaluated our novel NRA solver GANRA on two different benchmarks and demonstrated that using GPUs can significantly improve the speed of detecting potential satisfiable assignments for NRA problems. To go beyond the benefit of batching, "grouping" of similar operations can be employed to gain further speedups. While a manual implementation and optimization of each benchmark yields the highest performance, we have demonstrated that LLMs are capable of identifying underlying patterns and exploiting them to write efficient Python code themselves, removing the need for manual intervention.

ACKNOWLEDGMENTS

We want to thank the anonymous reviewers for their valuable feedback. We also want to thank Enrico Lipparini for his help in understanding UGOTNL, insights into the current state of the art, and positioning GANRA in this research landscape. Computations were performed with computing resources granted by RWTH Aachen University under project rwth1739 and p0023175.

REFERENCES

Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 13243, pp. 415–442. Springer International Publishing, 2022. doi: 10.1007/978-3-030-99524-9_24.

Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

Y.J. Cho and Y.Q. Chen. *Topological Degree Theory and Applications*. Mathematical Analysis and Applications. CRC Press, 2006. ISBN 9781420011487. URL https://books.google.de/books?id=6R6kpqHV_bAC.

Alessandro Cimatti, Alberto Griggio, Enrico Lipparini, and Roberto Sebastiani. Handling polynomial and transcendental functions in SMT via unconstrained optimisation and topological degree test. In Ahmed Bouajjani, Lukáš Holík, and Zhilin Wu (eds.), *Automated Technology for Verification and Analysis*, pp. 137–153, Cham, 2022. Springer International Publishing. ISBN 978-3-031-19992-9.

George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages*, pp. 134–183. Springer, 1975. doi: 10.1007/3-540-07407-4_17.

Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving. In *Theory and Applications of Satisfiability Testing – SAT 2015*, pp. 360–368. Springer International Publishing, 2015. doi: 10.1007/978-3-319-24318-4_26.

Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.

Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real Quantifier Elimination in Practice. In *Algorithmic Algebra and Number Theory*, pp. 221–247. Springer, 1999. doi: 10.1007/978-3-642-59932-3_11.

Bruno Dutertre. Yices 2.2. In *Computer Aided Verification*, pp. 737–744. Springer International Publishing, 2014. doi: 10.1007/978-3-319-08867-9_49.

Peter Franek and Stefan Ratschan. Effective topological degree computation based on interval arithmetic. *Mathematics of Computation*, 84, 07 2012. doi: 10.1090/S0025-5718-2014-02877-9.

Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina. Search-Space Partitioning for Parallelizing SMT Solvers. In *Theory and Applications of Satisfiability Testing – SAT 2015*, pp. 369–386, Cham, 2015. Springer International Publishing. doi: 10.1007/978-3-319-24318-4_27.

Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. *ACM Commun. Comput. Algebra*, 46(3/4):104–105, January 2013. ISSN 1932-2232. doi: 10.1145/2429135.2429155. URL https://doi.org/10.1145/2429135.2429155.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Enrico Lipparini. *Satisfiability modulo Nonlinear Arithmetic and Transcendental Functions via Numerical and Topological methods*. PhD thesis, University of Genova, 2024.

Enrico Lipparini and Stefan Ratschan. Satisfiability of non-linear transcendental arithmetic as a certificate search problem. *Journal of Automated Reasoning*, 69(1):3, 2025.

Minghao Liu, Kunhang Lv, Pei Huang, Rui Han, Fuqi Jia, Yu Zhang, Feifei Ma, and Jian Zhang. NRAgo: Solving SMT(NRA) Formulas with Gradient-Based Optimization. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 2046–2049, 2023. doi: 10.1109/ASE56229.2023.00013.

Matteo Marescotti, Antti Hyvärinen, and Natasha Sharygina. SMTS: Distributed, Visualized Constraint Solving. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pp. 534–524, 2018. doi: 10.29007/fhgn.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Muhammad Osama and Anton Wijs. GPU Acceleration of Bounded Model Checking with ParaFROST. In *Computer Aided Verification*, pp. 447–460, Cham, 2021. Springer International Publishing. doi: 10.1007/978-3-030-81688-9_21.

Muhammad Osama, Anton Wijs, and Armin Biere. Certified SAT solving with GPU accelerated inprocessing. *Formal Methods in System Design*, 62:79–118, June 2024. doi: 10.1007/s10703-023-00432-z.

Gilbert Stengle. A nullstellensatz and a positivstellensatz in semialgebraic geometry. 207:87–97, 1974. doi: 10.1007/BF01362149.

Thomas Sturm and Andreas Weber. Investigating generic methods to solve Hopf bifurcation problems in algebraic biology. In Katsuhisa Horimoto, Georg Regensburger, Markus Rosenkranz, and Hiroshi Yoshida (eds.), *Algebraic Biology*, pp. 200–215, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-85101-1.

David J. Wales and Jonathan P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997. doi: 10.1021/jp970984n. URL https://doi.org/10.1021/jp970984n.

Volker Weispfenning. The complexity of linear problems in fields. 5:3–27, 1988. doi: 10.1016/S0747-7171(88)80003-8.

Volker Weispfenning. Quantifier Elimination for Real Algebra — the Quadratic Case and Beyond. 8:85–101, 1997. doi: 10.1007/s002000050055.

Amalee Wilson, Andres Noetzli, Andrew Reynolds, Byron Cook, Cesare Tinelli, and Clark Barrett. Partitioning Strategies for Distributed SMT Solving. In *#PLACEHOLDER_PARENT_METADATA_VALUE#*, pp. 199–208. TU Wien Academic Press, October 2023. doi: 10.34727/2023/isbn.978-3-85448-060-0_28.

Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. A Concurrent Portfolio Approach to SMT Solving. In *Computer Aided Verification*, pp. 715–720, Berlin, Heidelberg, 2009. Springer. doi: 10.1007/978-3-642-02658-4_60.

Mengyu Zhao, Shaowei Cai, and Yuhang Qian. Distributed SMT Solving Based on Dynamic Variable-Level Partitioning. In *Computer Aided Verification*, pp. 68–88, Cham, 2024. Springer Nature Switzerland. doi: 10.1007/978-3-031-65627-9_4.

	Z3		CVC5		UGOTNL		NRAgo	
	Orig	Interval	Orig	Interval	Orig	Interval	Orig	Interval
SAT [↑]	1	0	0	0	10	9	6	2

Table 3: Performance of baseline tools on the original vs. a modified version of the Sturm-MBO benchmark.

A Appendix

A.1 Custom MBO Benchmark Generation Details

For each instance, we define a fixed N (see Section 4.2) and uniformly sample which of the six $h_1, \dots, h_6$ to use with which exponents. The variable j_2 is multiplied only with a subset of products: We flip a weighted coin and multiply by j_2^k in 40% of the cases, for a uniformly sampled $k \in \{1, \dots, N\}$. Finally, we uniformly sample the scalar factors of each product from the integers 1 to 20, and assign a negative sign to them in 20% of the cases.

A.2 Tool Performances on the Original vs. Modified Sturm-MBO Benchmark

Table 3 compares the performances of all baseline tools on the original version of the Sturm-MBO benchmark vs. the modified version. In the modified version ("Interval"), we replace the search for a root of the polynomial with the search for assignments that yield a positive (negative) result. If both assignments can be found, the existence of the root has been shown. Interval iteration could be used to find tighter bounds on this input.

All tools perform similar or slightly worse on the interval version of the benchmark, so in the main paper, we always report their performance on the original benchmark and only run GANRA on the modified version.

A.3 LLM Code Generation Prompts

Here, we provide the inputs and outputs to and from the LLM verbatim. Note that the initial prompt is identical for both benchmarks, except for the two provided examples. The samples were chosen to be small enough not to exceed the LLM character limit, but large enough to clearly demonstrate the underlying patterns. Future work could evaluate how to determine the ideal example instances.

A.3.1 Kissing

Initial Query

```
You are an expert at writing highly efficient PyTorch code.

Please write efficient Python code using PyTorch to solve the
following problem:
As an input, you receive a list of `K` formulas. Each formula is
given as a nested list that represents a tree. The first element
of inner nodes is the operation ("+", "-", "*" or "/") and which
is supposed to be applied to the following elements. The leafs
are strings or floating point numbers. Strings refer to variable
names. You also receive an input tensor `x` of shape `[batch_size,
N]`, where `N` is the number of variables across all formulas.
Finally, you receive a list `inputs` that contains the names of
all variables across all formulas. The order of entries maps the
variable names to their indices in `x`.
As an output, you should return a tensor of shape `[batch_size,
K]`, with the corresponding results for each formula.
```

```
Important: Your computation must be highly efficient! The code
can utilize a GPU, so you should use this to parallelize it
as much as possible. You must not use for-loops for any tensor
related operation. Instead, use tensor operations to perform all
required operations in parallel. Do not rely on JIT compilation
for speedups, but detect patterns in the operations so you can use
matrix-matrix operations and other torch features to parallelize
them. If formulas depend on multiple copies of some sub-term `a`,
do not recompute `a` for each instance. Instead, compute it only
once and then use broadcasting or advanced indexing to duplicate
the result.
Try to merge the operations in different branches of the tree: If
multiple branches perform the same type of operation, compute them
in parallel. E.g. instead of writing `a = x[:, 0]; b = x[:, 1];
a_squared = a ** 2; b_squared = b ** 2`, write `squared = x[:, :2]
** 2; a_squared = squared[:, 0]; b_squared = squared[:, 1]`. This
merges the squaring of `a` and `b`.

Your code doesn't have to parse arbitrary input formulas. Below,
you will see an example input. Extrapolate from this to detect
the underlying patterns. You do not have to account for any kind
of input that doesn't exhibit the same patterns. However, your
code must not hard code any of these values, but be usable for any
other input with similar structure.

Example:
[
['-', ['+', ['*', 'x_0_0', 'x_0_0'], ['+', ['*', 'x_0_1',
'x_0_1'], ['+', ['*', 'x_0_2', 'x_0_2'], ['*', 'x_0_3',
'x_0_3']]]], 1.0],
['-', ['+', ['*', 'x_1_0', 'x_1_0'], ['+', ['*', 'x_1_1',
'x_1_1'], ['+', ['*', 'x_1_2', 'x_1_2'], ['*', 'x_1_3',
'x_1_3']]]], 1.0],
['-', ['+', ['*', 'x_2_0', 'x_2_0'], ['+', ['*', 'x_2_1',
'x_2_1'], ['+', ['*', 'x_2_2', 'x_2_2'], ['*', 'x_2_3',
'x_2_3']]]], 1.0],
['-', ['+', ['*', 'x_3_0', 'x_3_0'], ['+', ['*', 'x_3_1',
'x_3_1'], ['+', ['*', 'x_3_2', 'x_3_2'], ['*', 'x_3_3',
'x_3_3']]]], 1.0],
['-', ['+', ['*', 'x_4_0', 'x_4_0'], ['+', ['*', 'x_4_1',
'x_4_1'], ['+', ['*', 'x_4_2', 'x_4_2'], ['*', 'x_4_3',
'x_4_3']]]], 1.0],
['-', ['+', ['*', 'x_5_0', 'x_5_0'], ['+', ['*', 'x_5_1',
'x_5_1'], ['+', ['*', 'x_5_2', 'x_5_2'], ['*', 'x_5_3',
'x_5_3']]]], 1.0],
['-', ['+', ['*', 'x_6_0', 'x_6_0'], ['+', ['*', 'x_6_1',
'x_6_1'], ['+', ['*', 'x_6_2', 'x_6_2'], ['*', 'x_6_3',
'x_6_3']]]], 1.0],
['-', ['+', ['*', 'x_7_0', 'x_7_0'], ['+', ['*', 'x_7_1',
'x_7_1'], ['+', ['*', 'x_7_2', 'x_7_2'], ['*', 'x_7_3',
'x_7_3']]]], 1.0],
['-', ['+', ['*', 'x_8_0', 'x_8_0'], ['+', ['*', 'x_8_1',
'x_8_1'], ['+', ['*', 'x_8_2', 'x_8_2'], ['*', 'x_8_3',
'x_8_3']]]], 1.0],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_1_0'], ['-', 'x_0_0',
'x_1_0']], ['+', ['*', ['-', 'x_0_1', 'x_1_1'], ['-', 'x_0_1',
'x_1_1']], ['+', ['*', ['-', 'x_0_2', 'x_1_2'], ['-', 'x_0_2',
'x_1_2']], ['*', ['-', 'x_0_3', 'x_1_3'], ['-', 'x_0_3',
'x_1_3']]]]]],
```

```
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_2_0'], ['-', 'x_0_0',
'x_2_0']], ['+', ['*', ['-', 'x_0_1', 'x_2_1'], ['-', 'x_0_1',
'x_2_1']], ['+', ['*', ['-', 'x_0_2', 'x_2_2'], ['-', 'x_0_2',
'x_2_2']], ['*', ['-', 'x_0_3', 'x_2_3'], ['-', 'x_0_3',
'x_2_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_3_0'], ['-', 'x_0_0',
'x_3_0']], ['+', ['*', ['-', 'x_0_1', 'x_3_1'], ['-', 'x_0_1',
'x_3_1']], ['+', ['*', ['-', 'x_0_2', 'x_3_2'], ['-', 'x_0_2',
'x_3_2']], ['*', ['-', 'x_0_3', 'x_3_3'], ['-', 'x_0_3',
'x_3_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_4_0'], ['-', 'x_0_0',
'x_4_0']], ['+', ['*', ['-', 'x_0_1', 'x_4_1'], ['-', 'x_0_1',
'x_4_1']], ['+', ['*', ['-', 'x_0_2', 'x_4_2'], ['-', 'x_0_2',
'x_4_2']], ['*', ['-', 'x_0_3', 'x_4_3'], ['-', 'x_0_3',
'x_4_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_5_0'], ['-', 'x_0_0',
'x_5_0']], ['+', ['*', ['-', 'x_0_1', 'x_5_1'], ['-', 'x_0_1',
'x_5_1']], ['+', ['*', ['-', 'x_0_2', 'x_5_2'], ['-', 'x_0_2',
'x_5_2']], ['*', ['-', 'x_0_3', 'x_5_3'], ['-', 'x_0_3',
'x_5_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_6_0'], ['-', 'x_0_0',
'x_6_0']], ['+', ['*', ['-', 'x_0_1', 'x_6_1'], ['-', 'x_0_1',
'x_6_1']], ['+', ['*', ['-', 'x_0_2', 'x_6_2'], ['-', 'x_0_2',
'x_6_2']], ['*', ['-', 'x_0_3', 'x_6_3'], ['-', 'x_0_3',
'x_6_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_7_0'], ['-', 'x_0_0',
'x_7_0']], ['+', ['*', ['-', 'x_0_1', 'x_7_1'], ['-', 'x_0_1',
'x_7_1']], ['+', ['*', ['-', 'x_0_2', 'x_7_2'], ['-', 'x_0_2',
'x_7_2']], ['*', ['-', 'x_0_3', 'x_7_3'], ['-', 'x_0_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_8_0'], ['-', 'x_0_0',
'x_8_0']], ['+', ['*', ['-', 'x_0_1', 'x_8_1'], ['-', 'x_0_1',
'x_8_1']], ['+', ['*', ['-', 'x_0_2', 'x_8_2'], ['-', 'x_0_2',
'x_8_2']], ['*', ['-', 'x_0_3', 'x_8_3'], ['-', 'x_0_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_2_0'], ['-', 'x_1_0',
'x_2_0']], ['+', ['*', ['-', 'x_1_1', 'x_2_1'], ['-', 'x_1_1',
'x_2_1']], ['+', ['*', ['-', 'x_1_2', 'x_2_2'], ['-', 'x_1_2',
'x_2_2']], ['*', ['-', 'x_1_3', 'x_2_3'], ['-', 'x_1_3',
'x_2_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_3_0'], ['-', 'x_1_0',
'x_3_0']], ['+', ['*', ['-', 'x_1_1', 'x_3_1'], ['-', 'x_1_1',
'x_3_1']], ['+', ['*', ['-', 'x_1_2', 'x_3_2'], ['-', 'x_1_2',
'x_3_2']], ['*', ['-', 'x_1_3', 'x_3_3'], ['-', 'x_1_3',
'x_3_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_4_0'], ['-', 'x_1_0',
'x_4_0']], ['+', ['*', ['-', 'x_1_1', 'x_4_1'], ['-', 'x_1_1',
'x_4_1']], ['+', ['*', ['-', 'x_1_2', 'x_4_2'], ['-', 'x_1_2',
'x_4_2']], ['*', ['-', 'x_1_3', 'x_4_3'], ['-', 'x_1_3',
'x_4_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_5_0'], ['-', 'x_1_0',
'x_5_0']], ['+', ['*', ['-', 'x_1_1', 'x_5_1'], ['-', 'x_1_1',
'x_5_1']], ['+', ['*', ['-', 'x_1_2', 'x_5_2'], ['-', 'x_1_2',
'x_5_2']], ['*', ['-', 'x_1_3', 'x_5_3'], ['-', 'x_1_3',
'x_5_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_6_0'], ['-', 'x_1_0',
'x_6_0']], ['+', ['*', ['-', 'x_1_1', 'x_6_1'], ['-', 'x_1_1',
'x_6_1']], ['+', ['*', ['-', 'x_1_2', 'x_6_2'], ['-', 'x_1_2',
```

```
'x_6_2']], ['*', ['-', 'x_1_3', 'x_6_3'], ['-', 'x_1_3',
'x_6_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_7_0'], ['-', 'x_1_0',
'x_7_0']], ['+', ['*', ['-', 'x_1_1', 'x_7_1'], ['-', 'x_1_1',
'x_7_1']], ['+', ['*', ['-', 'x_1_2', 'x_7_2'], ['-', 'x_1_2',
'x_7_2']], ['*', ['-', 'x_1_3', 'x_7_3'], ['-', 'x_1_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_8_0'], ['-', 'x_1_0',
'x_8_0']], ['+', ['*', ['-', 'x_1_1', 'x_8_1'], ['-', 'x_1_1',
'x_8_1']], ['+', ['*', ['-', 'x_1_2', 'x_8_2'], ['-', 'x_1_2',
'x_8_2']], ['*', ['-', 'x_1_3', 'x_8_3'], ['-', 'x_1_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_3_0'], ['-', 'x_2_0',
'x_3_0']], ['+', ['*', ['-', 'x_2_1', 'x_3_1'], ['-', 'x_2_1',
'x_3_1']], ['+', ['*', ['-', 'x_2_2', 'x_3_2'], ['-', 'x_2_2',
'x_3_2']], ['*', ['-', 'x_2_3', 'x_3_3'], ['-', 'x_2_3',
'x_3_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_4_0'], ['-', 'x_2_0',
'x_4_0']], ['+', ['*', ['-', 'x_2_1', 'x_4_1'], ['-', 'x_2_1',
'x_4_1']], ['+', ['*', ['-', 'x_2_2', 'x_4_2'], ['-', 'x_2_2',
'x_4_2']], ['*', ['-', 'x_2_3', 'x_4_3'], ['-', 'x_2_3',
'x_4_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_5_0'], ['-', 'x_2_0',
'x_5_0']], ['+', ['*', ['-', 'x_2_1', 'x_5_1'], ['-', 'x_2_1',
'x_5_1']], ['+', ['*', ['-', 'x_2_2', 'x_5_2'], ['-', 'x_2_2',
'x_5_2']], ['*', ['-', 'x_2_3', 'x_5_3'], ['-', 'x_2_3',
'x_5_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_6_0'], ['-', 'x_2_0',
'x_6_0']], ['+', ['*', ['-', 'x_2_1', 'x_6_1'], ['-', 'x_2_1',
'x_6_1']], ['+', ['*', ['-', 'x_2_2', 'x_6_2'], ['-', 'x_2_2',
'x_6_2']], ['*', ['-', 'x_2_3', 'x_6_3'], ['-', 'x_2_3',
'x_6_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_7_0'], ['-', 'x_2_0',
'x_7_0']], ['+', ['*', ['-', 'x_2_1', 'x_7_1'], ['-', 'x_2_1',
'x_7_1']], ['+', ['*', ['-', 'x_2_2', 'x_7_2'], ['-', 'x_2_2',
'x_7_2']], ['*', ['-', 'x_2_3', 'x_7_3'], ['-', 'x_2_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_8_0'], ['-', 'x_2_0',
'x_8_0']], ['+', ['*', ['-', 'x_2_1', 'x_8_1'], ['-', 'x_2_1',
'x_8_1']], ['+', ['*', ['-', 'x_2_2', 'x_8_2'], ['-', 'x_2_2',
'x_8_2']], ['*', ['-', 'x_2_3', 'x_8_3'], ['-', 'x_2_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_4_0'], ['-', 'x_3_0',
'x_4_0']], ['+', ['*', ['-', 'x_3_1', 'x_4_1'], ['-', 'x_3_1',
'x_4_1']], ['+', ['*', ['-', 'x_3_2', 'x_4_2'], ['-', 'x_3_2',
'x_4_2']], ['*', ['-', 'x_3_3', 'x_4_3'], ['-', 'x_3_3',
'x_4_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_5_0'], ['-', 'x_3_0',
'x_5_0']], ['+', ['*', ['-', 'x_3_1', 'x_5_1'], ['-', 'x_3_1',
'x_5_1']], ['+', ['*', ['-', 'x_3_2', 'x_5_2'], ['-', 'x_3_2',
'x_5_2']], ['*', ['-', 'x_3_3', 'x_5_3'], ['-', 'x_3_3',
'x_5_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_6_0'], ['-', 'x_3_0',
'x_6_0']], ['+', ['*', ['-', 'x_3_1', 'x_6_1'], ['-', 'x_3_1',
'x_6_1']], ['+', ['*', ['-', 'x_3_2', 'x_6_2'], ['-', 'x_3_2',
'x_6_2']], ['*', ['-', 'x_3_3', 'x_6_3'], ['-', 'x_3_3',
'x_6_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_7_0'], ['-', 'x_3_0',
'x_7_0']], ['+', ['*', ['-', 'x_3_1', 'x_7_1'], ['-', 'x_3_1',
```

```
'x_7_1']], ['+', ['*', ['-', 'x_3_2', 'x_7_2'], ['-', 'x_3_2',
'x_7_2']], ['*', ['-', 'x_3_3', 'x_7_3'], ['-', 'x_3_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_8_0'], ['-', 'x_3_0',
'x_8_0']], ['+', ['*', ['-', 'x_3_1', 'x_8_1'], ['-', 'x_3_1',
'x_8_1']], ['+', ['*', ['-', 'x_3_2', 'x_8_2'], ['-', 'x_3_2',
'x_8_2']], ['*', ['-', 'x_3_3', 'x_8_3'], ['-', 'x_3_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_5_0'], ['-', 'x_4_0',
'x_5_0']], ['+', ['*', ['-', 'x_4_1', 'x_5_1'], ['-', 'x_4_1',
'x_5_1']], ['+', ['*', ['-', 'x_4_2', 'x_5_2'], ['-', 'x_4_2',
'x_5_2']], ['*', ['-', 'x_4_3', 'x_5_3'], ['-', 'x_4_3',
'x_5_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_6_0'], ['-', 'x_4_0',
'x_6_0']], ['+', ['*', ['-', 'x_4_1', 'x_6_1'], ['-', 'x_4_1',
'x_6_1']], ['+', ['*', ['-', 'x_4_2', 'x_6_2'], ['-', 'x_4_2',
'x_6_2']], ['*', ['-', 'x_4_3', 'x_6_3'], ['-', 'x_4_3',
'x_6_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_7_0'], ['-', 'x_4_0',
'x_7_0']], ['+', ['*', ['-', 'x_4_1', 'x_7_1'], ['-', 'x_4_1',
'x_7_1']], ['+', ['*', ['-', 'x_4_2', 'x_7_2'], ['-', 'x_4_2',
'x_7_2']], ['*', ['-', 'x_4_3', 'x_7_3'], ['-', 'x_4_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_8_0'], ['-', 'x_4_0',
'x_8_0']], ['+', ['*', ['-', 'x_4_1', 'x_8_1'], ['-', 'x_4_1',
'x_8_1']], ['+', ['*', ['-', 'x_4_2', 'x_8_2'], ['-', 'x_4_2',
'x_8_2']], ['*', ['-', 'x_4_3', 'x_8_3'], ['-', 'x_4_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_6_0'], ['-', 'x_5_0',
'x_6_0']], ['+', ['*', ['-', 'x_5_1', 'x_6_1'], ['-', 'x_5_1',
'x_6_1']], ['+', ['*', ['-', 'x_5_2', 'x_6_2'], ['-', 'x_5_2',
'x_6_2']], ['*', ['-', 'x_5_3', 'x_6_3'], ['-', 'x_5_3',
'x_6_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_7_0'], ['-', 'x_5_0',
'x_7_0']], ['+', ['*', ['-', 'x_5_1', 'x_7_1'], ['-', 'x_5_1',
'x_7_1']], ['+', ['*', ['-', 'x_5_2', 'x_7_2'], ['-', 'x_5_2',
'x_7_2']], ['*', ['-', 'x_5_3', 'x_7_3'], ['-', 'x_5_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_8_0'], ['-', 'x_5_0',
'x_8_0']], ['+', ['*', ['-', 'x_5_1', 'x_8_1'], ['-', 'x_5_1',
'x_8_1']], ['+', ['*', ['-', 'x_5_2', 'x_8_2'], ['-', 'x_5_2',
'x_8_2']], ['*', ['-', 'x_5_3', 'x_8_3'], ['-', 'x_5_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_6_0', 'x_7_0'], ['-', 'x_6_0',
'x_7_0']], ['+', ['*', ['-', 'x_6_1', 'x_7_1'], ['-', 'x_6_1',
'x_7_1']], ['+', ['*', ['-', 'x_6_2', 'x_7_2'], ['-', 'x_6_2',
'x_7_2']], ['*', ['-', 'x_6_3', 'x_7_3'], ['-', 'x_6_3',
'x_7_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_6_0', 'x_8_0'], ['-', 'x_6_0',
'x_8_0']], ['+', ['*', ['-', 'x_6_1', 'x_8_1'], ['-', 'x_6_1',
'x_8_1']], ['+', ['*', ['-', 'x_6_2', 'x_8_2'], ['-', 'x_6_2',
'x_8_2']], ['*', ['-', 'x_6_3', 'x_8_3'], ['-', 'x_6_3',
'x_8_3']]]]]],
['-', 1.0, ['+', ['*', ['-', 'x_7_0', 'x_8_0'], ['-', 'x_7_0',
'x_8_0']], ['+', ['*', ['-', 'x_7_1', 'x_8_1'], ['-', 'x_7_1',
'x_8_1']], ['+', ['*', ['-', 'x_7_2', 'x_8_2'], ['-', 'x_7_2',
'x_8_2']], ['*', ['-', 'x_7_3', 'x_8_3'], ['-', 'x_7_3',
'x_8_3']]]]]]
]
```

```
Another example:
[
['-', ['+', ['*', 'x_0_0', 'x_0_0'], ['+', ['*', 'x_0_1',
'x_0_1'], ['*', 'x_0_2', 'x_0_2']]], 1.0],
['-', ['+', ['*', 'x_1_0', 'x_1_0'], ['+', ['*', 'x_1_1',
'x_1_1'], ['*', 'x_1_2', 'x_1_2']]], 1.0],
['-', ['+', ['*', 'x_2_0', 'x_2_0'], ['+', ['*', 'x_2_1',
'x_2_1'], ['*', 'x_2_2', 'x_2_2']]], 1.0],
['-', ['+', ['*', 'x_3_0', 'x_3_0'], ['+', ['*', 'x_3_1',
'x_3_1'], ['*', 'x_3_2', 'x_3_2']]], 1.0],
['-', ['+', ['*', 'x_4_0', 'x_4_0'], ['+', ['*', 'x_4_1',
'x_4_1'], ['*', 'x_4_2', 'x_4_2']]], 1.0],
['-', ['+', ['*', 'x_5_0', 'x_5_0'], ['+', ['*', 'x_5_1',
'x_5_1'], ['*', 'x_5_2', 'x_5_2']]], 1.0],
['-', ['+', ['*', 'x_6_0', 'x_6_0'], ['+', ['*', 'x_6_1',
'x_6_1'], ['*', 'x_6_2', 'x_6_2']]], 1.0],
['-', ['+', ['*', 'x_7_0', 'x_7_0'], ['+', ['*', 'x_7_1',
'x_7_1'], ['*', 'x_7_2', 'x_7_2']]], 1.0],
['-', ['+', ['*', 'x_8_0', 'x_8_0'], ['+', ['*', 'x_8_1',
'x_8_1'], ['*', 'x_8_2', 'x_8_2']]], 1.0],
['-', ['+', ['*', 'x_9_0', 'x_9_0'], ['+', ['*', 'x_9_1',
'x_9_1'], ['*', 'x_9_2', 'x_9_2']]], 1.0],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_1_0'], ['-', 'x_0_0',
'x_1_0']], ['+', ['*', ['-', 'x_0_1', 'x_1_1'], ['-', 'x_0_1',
'x_1_1']], ['*', ['-', 'x_0_2', 'x_1_2'], ['-', 'x_0_2',
'x_1_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_2_0'], ['-', 'x_0_0',
'x_2_0']], ['+', ['*', ['-', 'x_0_1', 'x_2_1'], ['-', 'x_0_1',
'x_2_1']], ['*', ['-', 'x_0_2', 'x_2_2'], ['-', 'x_0_2',
'x_2_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_3_0'], ['-', 'x_0_0',
'x_3_0']], ['+', ['*', ['-', 'x_0_1', 'x_3_1'], ['-', 'x_0_1',
'x_3_1']], ['*', ['-', 'x_0_2', 'x_3_2'], ['-', 'x_0_2',
'x_3_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_4_0'], ['-', 'x_0_0',
'x_4_0']], ['+', ['*', ['-', 'x_0_1', 'x_4_1'], ['-', 'x_0_1',
'x_4_1']], ['*', ['-', 'x_0_2', 'x_4_2'], ['-', 'x_0_2',
'x_4_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_5_0'], ['-', 'x_0_0',
'x_5_0']], ['+', ['*', ['-', 'x_0_1', 'x_5_1'], ['-', 'x_0_1',
'x_5_1']], ['*', ['-', 'x_0_2', 'x_5_2'], ['-', 'x_0_2',
'x_5_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_6_0'], ['-', 'x_0_0',
'x_6_0']], ['+', ['*', ['-', 'x_0_1', 'x_6_1'], ['-', 'x_0_1',
'x_6_1']], ['*', ['-', 'x_0_2', 'x_6_2'], ['-', 'x_0_2',
'x_6_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_7_0'], ['-', 'x_0_0',
'x_7_0']], ['+', ['*', ['-', 'x_0_1', 'x_7_1'], ['-', 'x_0_1',
'x_7_1']], ['*', ['-', 'x_0_2', 'x_7_2'], ['-', 'x_0_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_8_0'], ['-', 'x_0_0',
'x_8_0']], ['+', ['*', ['-', 'x_0_1', 'x_8_1'], ['-', 'x_0_1',
'x_8_1']], ['*', ['-', 'x_0_2', 'x_8_2'], ['-', 'x_0_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_0_0', 'x_9_0'], ['-', 'x_0_0',
'x_9_0']], ['+', ['*', ['-', 'x_0_1', 'x_9_1'], ['-', 'x_0_1',
'x_9_1']], ['*', ['-', 'x_0_2', 'x_9_2'], ['-', 'x_0_2',
'x_9_2']]]]],
```

```
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_2_0'], ['-', 'x_1_0',
'x_2_0']], ['+', ['*', ['-', 'x_1_1', 'x_2_1'], ['-', 'x_1_1',
'x_2_1']], ['*', ['-', 'x_1_2', 'x_2_2'], ['-', 'x_1_2',
'x_2_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_3_0'], ['-', 'x_1_0',
'x_3_0']], ['+', ['*', ['-', 'x_1_1', 'x_3_1'], ['-', 'x_1_1',
'x_3_1']], ['*', ['-', 'x_1_2', 'x_3_2'], ['-', 'x_1_2',
'x_3_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_4_0'], ['-', 'x_1_0',
'x_4_0']], ['+', ['*', ['-', 'x_1_1', 'x_4_1'], ['-', 'x_1_1',
'x_4_1']], ['*', ['-', 'x_1_2', 'x_4_2'], ['-', 'x_1_2',
'x_4_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_5_0'], ['-', 'x_1_0',
'x_5_0']], ['+', ['*', ['-', 'x_1_1', 'x_5_1'], ['-', 'x_1_1',
'x_5_1']], ['*', ['-', 'x_1_2', 'x_5_2'], ['-', 'x_1_2',
'x_5_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_6_0'], ['-', 'x_1_0',
'x_6_0']], ['+', ['*', ['-', 'x_1_1', 'x_6_1'], ['-', 'x_1_1',
'x_6_1']], ['*', ['-', 'x_1_2', 'x_6_2'], ['-', 'x_1_2',
'x_6_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_7_0'], ['-', 'x_1_0',
'x_7_0']], ['+', ['*', ['-', 'x_1_1', 'x_7_1'], ['-', 'x_1_1',
'x_7_1']], ['*', ['-', 'x_1_2', 'x_7_2'], ['-', 'x_1_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_8_0'], ['-', 'x_1_0',
'x_8_0']], ['+', ['*', ['-', 'x_1_1', 'x_8_1'], ['-', 'x_1_1',
'x_8_1']], ['*', ['-', 'x_1_2', 'x_8_2'], ['-', 'x_1_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_1_0', 'x_9_0'], ['-', 'x_1_0',
'x_9_0']], ['+', ['*', ['-', 'x_1_1', 'x_9_1'], ['-', 'x_1_1',
'x_9_1']], ['*', ['-', 'x_1_2', 'x_9_2'], ['-', 'x_1_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_3_0'], ['-', 'x_2_0',
'x_3_0']], ['+', ['*', ['-', 'x_2_1', 'x_3_1'], ['-', 'x_2_1',
'x_3_1']], ['*', ['-', 'x_2_2', 'x_3_2'], ['-', 'x_2_2',
'x_3_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_4_0'], ['-', 'x_2_0',
'x_4_0']], ['+', ['*', ['-', 'x_2_1', 'x_4_1'], ['-', 'x_2_1',
'x_4_1']], ['*', ['-', 'x_2_2', 'x_4_2'], ['-', 'x_2_2',
'x_4_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_5_0'], ['-', 'x_2_0',
'x_5_0']], ['+', ['*', ['-', 'x_2_1', 'x_5_1'], ['-', 'x_2_1',
'x_5_1']], ['*', ['-', 'x_2_2', 'x_5_2'], ['-', 'x_2_2',
'x_5_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_6_0'], ['-', 'x_2_0',
'x_6_0']], ['+', ['*', ['-', 'x_2_1', 'x_6_1'], ['-', 'x_2_1',
'x_6_1']], ['*', ['-', 'x_2_2', 'x_6_2'], ['-', 'x_2_2',
'x_6_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_7_0'], ['-', 'x_2_0',
'x_7_0']], ['+', ['*', ['-', 'x_2_1', 'x_7_1'], ['-', 'x_2_1',
'x_7_1']], ['*', ['-', 'x_2_2', 'x_7_2'], ['-', 'x_2_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_8_0'], ['-', 'x_2_0',
'x_8_0']], ['+', ['*', ['-', 'x_2_1', 'x_8_1'], ['-', 'x_2_1',
'x_8_1']], ['*', ['-', 'x_2_2', 'x_8_2'], ['-', 'x_2_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_2_0', 'x_9_0'], ['-', 'x_2_0',
'x_9_0']], ['+', ['*', ['-', 'x_2_1', 'x_9_1'], ['-', 'x_2_1',
```

```
'x_9_1']], ['*', ['-', 'x_2_2', 'x_9_2'], ['-', 'x_2_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_4_0'], ['-', 'x_3_0',
'x_4_0']], ['+', ['*', ['-', 'x_3_1', 'x_4_1'], ['-', 'x_3_1',
'x_4_1']], ['*', ['-', 'x_3_2', 'x_4_2'], ['-', 'x_3_2',
'x_4_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_5_0'], ['-', 'x_3_0',
'x_5_0']], ['+', ['*', ['-', 'x_3_1', 'x_5_1'], ['-', 'x_3_1',
'x_5_1']], ['*', ['-', 'x_3_2', 'x_5_2'], ['-', 'x_3_2',
'x_5_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_6_0'], ['-', 'x_3_0',
'x_6_0']], ['+', ['*', ['-', 'x_3_1', 'x_6_1'], ['-', 'x_3_1',
'x_6_1']], ['*', ['-', 'x_3_2', 'x_6_2'], ['-', 'x_3_2',
'x_6_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_7_0'], ['-', 'x_3_0',
'x_7_0']], ['+', ['*', ['-', 'x_3_1', 'x_7_1'], ['-', 'x_3_1',
'x_7_1']], ['*', ['-', 'x_3_2', 'x_7_2'], ['-', 'x_3_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_8_0'], ['-', 'x_3_0',
'x_8_0']], ['+', ['*', ['-', 'x_3_1', 'x_8_1'], ['-', 'x_3_1',
'x_8_1']], ['*', ['-', 'x_3_2', 'x_8_2'], ['-', 'x_3_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_3_0', 'x_9_0'], ['-', 'x_3_0',
'x_9_0']], ['+', ['*', ['-', 'x_3_1', 'x_9_1'], ['-', 'x_3_1',
'x_9_1']], ['*', ['-', 'x_3_2', 'x_9_2'], ['-', 'x_3_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_5_0'], ['-', 'x_4_0',
'x_5_0']], ['+', ['*', ['-', 'x_4_1', 'x_5_1'], ['-', 'x_4_1',
'x_5_1']], ['*', ['-', 'x_4_2', 'x_5_2'], ['-', 'x_4_2',
'x_5_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_6_0'], ['-', 'x_4_0',
'x_6_0']], ['+', ['*', ['-', 'x_4_1', 'x_6_1'], ['-', 'x_4_1',
'x_6_1']], ['*', ['-', 'x_4_2', 'x_6_2'], ['-', 'x_4_2',
'x_6_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_7_0'], ['-', 'x_4_0',
'x_7_0']], ['+', ['*', ['-', 'x_4_1', 'x_7_1'], ['-', 'x_4_1',
'x_7_1']], ['*', ['-', 'x_4_2', 'x_7_2'], ['-', 'x_4_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_8_0'], ['-', 'x_4_0',
'x_8_0']], ['+', ['*', ['-', 'x_4_1', 'x_8_1'], ['-', 'x_4_1',
'x_8_1']], ['*', ['-', 'x_4_2', 'x_8_2'], ['-', 'x_4_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_4_0', 'x_9_0'], ['-', 'x_4_0',
'x_9_0']], ['+', ['*', ['-', 'x_4_1', 'x_9_1'], ['-', 'x_4_1',
'x_9_1']], ['*', ['-', 'x_4_2', 'x_9_2'], ['-', 'x_4_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_6_0'], ['-', 'x_5_0',
'x_6_0']], ['+', ['*', ['-', 'x_5_1', 'x_6_1'], ['-', 'x_5_1',
'x_6_1']], ['*', ['-', 'x_5_2', 'x_6_2'], ['-', 'x_5_2',
'x_6_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_7_0'], ['-', 'x_5_0',
'x_7_0']], ['+', ['*', ['-', 'x_5_1', 'x_7_1'], ['-', 'x_5_1',
'x_7_1']], ['*', ['-', 'x_5_2', 'x_7_2'], ['-', 'x_5_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_8_0'], ['-', 'x_5_0',
'x_8_0']], ['+', ['*', ['-', 'x_5_1', 'x_8_1'], ['-', 'x_5_1',
'x_8_1']], ['*', ['-', 'x_5_2', 'x_8_2'], ['-', 'x_5_2',
'x_8_2']]]]],
```

```
['-', 1.0, ['+', ['*', ['-', 'x_5_0', 'x_9_0'], ['-', 'x_5_0',
'x_9_0']], ['+', ['*', ['-', 'x_5_1', 'x_9_1'], ['-', 'x_5_1',
'x_9_1']], ['*', ['-', 'x_5_2', 'x_9_2'], ['-', 'x_5_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_6_0', 'x_7_0'], ['-', 'x_6_0',
'x_7_0']], ['+', ['*', ['-', 'x_6_1', 'x_7_1'], ['-', 'x_6_1',
'x_7_1']], ['*', ['-', 'x_6_2', 'x_7_2'], ['-', 'x_6_2',
'x_7_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_6_0', 'x_8_0'], ['-', 'x_6_0',
'x_8_0']], ['+', ['*', ['-', 'x_6_1', 'x_8_1'], ['-', 'x_6_1',
'x_8_1']], ['*', ['-', 'x_6_2', 'x_8_2'], ['-', 'x_6_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_6_0', 'x_9_0'], ['-', 'x_6_0',
'x_9_0']], ['+', ['*', ['-', 'x_6_1', 'x_9_1'], ['-', 'x_6_1',
'x_9_1']], ['*', ['-', 'x_6_2', 'x_9_2'], ['-', 'x_6_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_7_0', 'x_8_0'], ['-', 'x_7_0',
'x_8_0']], ['+', ['*', ['-', 'x_7_1', 'x_8_1'], ['-', 'x_7_1',
'x_8_1']], ['*', ['-', 'x_7_2', 'x_8_2'], ['-', 'x_7_2',
'x_8_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_7_0', 'x_9_0'], ['-', 'x_7_0',
'x_9_0']], ['+', ['*', ['-', 'x_7_1', 'x_9_1'], ['-', 'x_7_1',
'x_9_1']], ['*', ['-', 'x_7_2', 'x_9_2'], ['-', 'x_7_2',
'x_9_2']]]]],
['-', 1.0, ['+', ['*', ['-', 'x_8_0', 'x_9_0'], ['-', 'x_8_0',
'x_9_0']], ['+', ['*', ['-', 'x_8_1', 'x_9_1'], ['-', 'x_8_1',
'x_9_1']], ['*', ['-', 'x_8_2', 'x_9_2'], ['-', 'x_8_2',
'x_9_2']]]]]
]

Perform the following steps: First, analyze the example and detect
the underlying patterns. Finally, implement the logic and provide
the python code. Your code must be able to deal with negative
variables, too. So do not use logarithms. Make sure to place all
tensors on the same device that the x tensor is on.

Write a function `compute_result(formulas, x, inputs)` that
receives the formulas, the variables x and the mapping to variable
names as an input, and returns a tensor with the corresponding
results.
```

LLM Response

```
Based on the patterns observed in the provided examples, the
formulas calculate two types of expressions:

1. For each unique index `i`, compute \( s_i = \sum_{k} x_{i,k}^2
- 1.0 \).
2. For each pair of unique indices `(i, j)`, compute \( s_{ij} =
1.0 - \sum_{k} (x_{i,k} - x_{j,k})^2 \).

Here's the efficient PyTorch implementation:

```python
import torch

def compute_result(formulas, x, inputs):
    device = x.device
    batch_size = x.shape[0]
```
```

```
    # Build mapping from variable names to indices in x
    variable_mapping = {}
    for idx, var_name in enumerate(inputs):
        tokens = var_name.split('_')
        if len(tokens) != 3 or not tokens[1].isdigit() or not
tokens[2].isdigit():
            continue  # Skip invalid variable names
        i = int(tokens[1])
        k = int(tokens[2])
        variable_mapping[var_name] = {'i': i, 'k': k,
'index_in_x': idx}

    # Map i to list of (k, idx_in_x)
    i_to_k_idx = {}
    for var_name, data in variable_mapping.items():
        i = data['i']
        k = data['k']
        idx_in_x = data['index_in_x']
        if i not in i_to_k_idx:
            i_to_k_idx[i] = []
        i_to_k_idx[i].append((k, idx_in_x))

    # Get sorted list of unique i's
    unique_is = sorted(i_to_k_idx.keys())
    num_i = len(unique_is)

    # Build x_i_list
    x_i_list = []
    for i in unique_is:
        k_idx_pairs = sorted(i_to_k_idx[i], key=lambda x: x[0])  #
Sort by k
        idxs_for_i = [idx_in_x for k, idx_in_x in k_idx_pairs]
        x_i = x[:, idxs_for_i]  # Shape [batch_size, D]
        x_i_list.append(x_i)

    # Stack x_i tensors
    X = torch.stack(x_i_list, dim=0).to(device)  # Shape [num_i,
batch_size, D]

    # Transpose to shape [batch_size, num_i, D]
    X = X.permute(1, 0, 2)  # Shape [batch_size, num_i, D]

    # Compute x_i squared norms
    x_i_squared_norms = (X ** 2).sum(dim=2)  # Shape [batch_size,
num_i]

    s_i = x_i_squared_norms - 1.0  # Shape [batch_size, num_i]

    # Compute pairwise squared distances
    x_i_xj_dot = torch.bmm(X, X.transpose(1,2))  # Shape
[batch_size, num_i, num_i]

    # Compute s_ij = 1.0 - ||x_i - x_j||^2
    s_ij = 1.0 - (
        x_i_squared_norms.unsqueeze(2) + x_i_squared_norms.unsqueeze(1)
- 2 * x_i_xj_dot
    )  # Shape [batch_size, num_i, num_i]
```

```
    # Extract upper triangle indices
    indices_i, indices_j = torch.triu_indices(num_i, num_i,
offset=1)

    # s_ij_upper has shape [batch_size, num_pairs]
    s_ij_upper = s_ij[:, indices_i, indices_j]

    # Concatenate s_i and s_ij_upper
    output = torch.cat([s_i, s_ij_upper], dim=1)  # Shape
[batch_size, K]

    return output
```

This function efficiently computes the required results by:

- Parsing the input variable names to map each unique `i` to its corresponding columns in `x`.
- Stacking the vectors \(x_i \) for each unique `i` to form a tensor.
- Computing \(s_i \) and \(s_{ij} \) using vectorized operations without any for-loops.
- Ensuring that all tensors are on the same device as `x`, and handling negative values correctly by avoiding square roots or logarithms.

This code should work efficiently for inputs with similar structures to the examples provided.

A.3.2 STURM-MBO

Initial Query

```
You are an expert at writing highly efficient PyTorch code.

Please write efficient Python code using PyTorch to solve the
following problem:
As an input, you receive a list of `K` formulas. Each formula is
given as a nested list that represents a tree. The first element
of inner nodes is the operation ("+", "-", "*" or "/") and which
is supposed to be applied to the following elements. The leafs
are strings or floating point numbers. Strings refer to variable
names. You also receive an input tensor `x` of shape `[batch_size,
N]`, where `N` is the number of variables across all formulas.
Finally, you receive a list `inputs` that contains the names of
all variables across all formulas. The order of entries maps the
variable names to their indices in `x`.
As an output, you should return a tensor of shape `[batch_size,
K]`, with the corresponding results for each formula.

Important: Your computation must be highly efficient! The code
can utilize a GPU, so you should use this to parallelize it
as much as possible. You must not use for-loops for any tensor
related operation. Instead, use tensor operations to perform all
required operations in parallel. Do not rely on JIT compilation
for speedups, but detect patterns in the operations so you can use
matrix-matrix operations and other torch features to parallelize
them. If formulas depend on multiple copies of some sub-term `a`,
do not recompute `a` for each instance. Instead, compute it only
```

```
once and then use broadcasting or advanced indexing to duplicate
the result.
Try to merge the operations in different branches of the tree: If
multiple branches perform the same type of operation, compute them
in parallel. E.g. instead of writing `a = x[:, 0]; b = x[:, 1];
a_squared = a ** 2; b_squared = b ** 2`, write `squared = x[:, :2]
** 2; a_squared = squared[:, 0]; b_squared = squared[:, 1]`. This
merges the squaring of `a` and `b`.

Your code doesn't have to parse arbitrary input formulas. Below,
you will see an example input. Extrapolate from this to detect
the underlying patterns. You do not have to account for any kind
of input that doesn't exhibit the same patterns. However, your
code must not hard code any of these values, but be usable for any
other input with similar structure.

Example:
[
['-', 0.0, ['+', ['*', -1.0, ['*', 2.0, ['*', 'h1p', 'h1p',
'h1p'], ['*', 'h2p', 'h2p'], 'h3p']], ['*', 2.0, ['*', 'h1p',
'h1p', 'h1p'], ['*', 'h2p', 'h2p'], 'h5p'], ['*', -1.0, ['*',
8.0, ['*', 'h1p', 'h1p', 'h1p'], 'h2p', ['*', 'h3p', 'h3p']]],
['*', 4.0, ['*', 'h1p', 'h1p', 'h1p'], 'h2p', 'h3p', 'h4p'], ['*',
10.0, ['*', 'h1p', 'h1p', 'h1p'], 'h2p', 'h3p', 'h5p'], ['*',
10.0, ['*', 'h1p', 'h1p', 'h1p'], 'h2p', 'h3p', 'h6p'], ['*', 2.0,
['*', 'h1p', 'h1p', 'h1p'], 'h2p', 'h4p', 'h5p'], ['*', 2.0, ['*',
'h1p', 'h1p', 'h1p'], 'h2p', ['*', 'h5p', 'h5p']], ['*', 24.0,
['*', 'h1p', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h4p'], ['*',
16.0, ['*', 'h1p', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h5p'],
['*', 16.0, ['*', 'h1p', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'],
'h6p'], ['*', 6.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p', ['*',
'h4p', 'h4p']], ['*', 6.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p',
'h4p', 'h5p'], ['*', 10.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p',
'h4p', 'h6p'], ['*', 4.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p', ['*',
'h5p', 'h5p']], ['*', 8.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p',
'h5p', 'h6p'], ['*', 4.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p',
['*', 'h6p', 'h6p']], ['*', 2.0, ['*', 'h1p', 'h1p', 'h1p'],
'h4p', 'h5p', 'h6p'], ['*', 2.0, ['*', 'h1p', 'h1p', 'h1p'],
['*', 'h5p', 'h5p'], 'h6p'], ['*', 2.0, ['*', 'h1p', 'h1p',
'h1p'], 'h5p', ['*', 'h6p', 'h6p']], ['*', -1.0, ['*', ['*',
'h1p', 'h1p'], ['*', 'h2p', 'h2p', 'h2p'], 'h3p']], ['*', ['*',
'h1p', 'h1p'], ['*', 'h2p', 'h2p', 'h2p'], 'h5p'], ['*', -1.0,
['*', 9.0, ['*', 'h1p', 'h1p'], ['*', 'h2p', 'h2p'], ['*', 'h3p',
'h3p']]], ['*', ['*', 'h1p', 'h1p'], ['*', 'h2p', 'h2p'], 'h3p',
'h4p'], ['*', 10.0, ['*', 'h1p', 'h1p'], ['*', 'h2p', 'h2p'],
'h3p', 'h5p'], ['*', 8.0, ['*', 'h1p', 'h1p'], ['*', 'h2p',
'h2p'], 'h3p', 'h6p'], ['*', 6.0, ['*', 'h1p', 'h1p'], ['*',
'h2p', 'h2p'], 'h4p', 'h5p'], ['*', 3.0, ['*', 'h1p', 'h1p'],
['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p']], ['*', 3.0, ['*', 'h1p',
'h1p'], ['*', 'h2p', 'h2p'], 'h5p', 'h6p'], ['*', -1.0, ['*',
8.0, ['*', 'h1p', 'h1p'], 'h2p', ['*', 'h3p', 'h3p', 'h3p']]],
['*', 28.0, ['*', 'h1p', 'h1p'], 'h2p', ['*', 'h3p', 'h3p'],
'h4p'], ['*', 26.0, ['*', 'h1p', 'h1p'], 'h2p', ['*', 'h3p',
'h3p'], 'h5p'], ['*', 34.0, ['*', 'h1p', 'h1p'], 'h2p', ['*',
'h3p', 'h3p'], 'h6p'], ['*', 13.0, ['*', 'h1p', 'h1p'], 'h2p',
'h3p', ['*', 'h4p', 'h4p']], ['*', 28.0, ['*', 'h1p', 'h1p'],
'h2p', 'h3p', 'h4p', 'h5p'], ['*', 30.0, ['*', 'h1p', 'h1p'],
'h2p', 'h3p', 'h4p', 'h6p'], ['*', 15.0, ['*', 'h1p', 'h1p'],
'h2p', 'h3p', ['*', 'h5p', 'h5p']], ['*', 32.0, ['*', 'h1p',
```

```
'h1p'], 'h2p', 'h3p', 'h5p', 'h6p'], ['*', 17.0, ['*', 'h1p',
'h1p'], 'h2p', 'h3p', ['*', 'h6p', 'h6p']], ['*', 3.0, ['*',
'h1p', 'h1p'], 'h2p', ['*', 'h4p', 'h4p'], 'h5p'], ['*', 6.0,
['*', 'h1p', 'h1p'], 'h2p', 'h4p', ['*', 'h5p', 'h5p']], ['*',
6.0, ['*', 'h1p', 'h1p'], 'h2p', 'h4p', 'h5p', 'h6p'], ['*', ['*',
'h1p', 'h1p'], 'h2p', ['*', 'h5p', 'h5p', 'h5p']], ['*', 4.0,
['*', 'h1p', 'h1p'], 'h2p', ['*', 'h5p', 'h5p'], 'h6p'], ['*',
3.0, ['*', 'h1p', 'h1p'], 'h2p', 'h5p', ['*', 'h6p', 'h6p']],
['*', 24.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p', 'h3p'],
'h4p'], ['*', 16.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p',
'h3p'], 'h5p'], ['*', 16.0, ['*', 'h1p', 'h1p'], ['*', 'h3p',
'h3p', 'h3p'], 'h6p'], ['*', 21.0, ['*', 'h1p', 'h1p'], ['*',
'h3p', 'h3p'], ['*', 'h4p', 'h4p']], ['*', 46.0, ['*', 'h1p',
'h1p'], ['*', 'h3p', 'h3p'], 'h4p', 'h5p'], ['*', 38.0, ['*',
'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h4p', 'h6p'], ['*', 16.0,
['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p']],
['*', 40.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h5p',
'h6p'], ['*', 16.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'],
['*', 'h6p', 'h6p']], ['*', 3.0, ['*', 'h1p', 'h1p'], 'h3p', ['*',
'h4p', 'h4p', 'h4p']], ['*', 14.0, ['*', 'h1p', 'h1p'], 'h3p',
['*', 'h4p', 'h4p'], 'h5p'], ['*', 8.0, ['*', 'h1p', 'h1p'],
'h3p', ['*', 'h4p', 'h4p'], 'h6p'], ['*', 11.0, ['*', 'h1p',
'h1p'], 'h3p', 'h4p', ['*', 'h5p', 'h5p']], ['*', 24.0, ['*',
'h1p', 'h1p'], 'h3p', 'h4p', 'h5p', 'h6p'], ['*', 7.0, ['*',
'h1p', 'h1p'], 'h3p', 'h4p', ['*', 'h6p', 'h6p']], ['*', 2.0,
['*', 'h1p', 'h1p'], 'h3p', ['*', 'h5p', 'h5p', 'h5p']], ['*',
12.0, ['*', 'h1p', 'h1p'], 'h3p', ['*', 'h5p', 'h5p'], 'h6p'],
['*', 12.0, ['*', 'h1p', 'h1p'], 'h3p', 'h5p', ['*', 'h6p',
'h6p']], ['*', 2.0, ['*', 'h1p', 'h1p'], 'h3p', ['*', 'h6p',
'h6p', 'h6p']], ['*', ['*', 'h1p', 'h1p'], ['*', 'h4p', 'h4p'],
'h5p', 'h6p'], ['*', 4.0, ['*', 'h1p', 'h1p'], 'h4p', ['*', 'h5p',
'h5p'], 'h6p'], ['*', 2.0, ['*', 'h1p', 'h1p'], 'h4p', 'h5p',
['*', 'h6p', 'h6p']], ['*', ['*', 'h1p', 'h1p'], ['*', 'h5p',
'h5p', 'h5p'], 'h6p'], ['*', 3.0, ['*', 'h1p', 'h1p'], ['*',
'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*', ['*', 'h1p', 'h1p'],
'h5p', ['*', 'h6p', 'h6p', 'h6p']], ['*', -1.0, ['*', 'h1p',
['*', 'h2p', 'h2p', 'h2p'], ['*', 'h3p', 'h3p']]], ['*', 'h1p',
['*', 'h2p', 'h2p', 'h2p'], 'h3p', 'h5p'], ['*', 2.0, 'h1p',
['*', 'h2p', 'h2p', 'h2p'], 'h3p', 'h6p'], ['*', 2.0, 'h1p',
['*', 'h2p', 'h2p', 'h2p'], 'h4p', 'h5p'], ['*', 'h1p', ['*',
'h2p', 'h2p', 'h2p'], ['*', 'h5p', 'h5p']], ['*', 'h1p', ['*',
'h2p', 'h2p', 'h2p'], 'h5p', 'h6p'], ['*', -1.0, ['*', 2.0, 'h1p',
['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p']]], ['*', 8.0,
'h1p', ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h4p'], ['*',
9.0, 'h1p', ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h5p'],
['*', 15.0, 'h1p', ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
'h6p'], ['*', 6.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h4p',
'h4p']], ['*', 20.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', 'h4p',
'h5p'], ['*', 15.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', 'h4p',
'h6p'], ['*', 8.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p',
'h5p']], ['*', 19.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', 'h5p',
'h6p'], ['*', 9.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h6p',
'h6p']], ['*', 5.0, 'h1p', ['*', 'h2p', 'h2p'], ['*', 'h4p',
'h4p'], 'h5p'], ['*', 5.0, 'h1p', ['*', 'h2p', 'h2p'], 'h4p',
['*', 'h5p', 'h5p']], ['*', 9.0, 'h1p', ['*', 'h2p', 'h2p'],
'h4p', 'h5p', 'h6p'], ['*', 'h1p', ['*', 'h2p', 'h2p'], ['*',
'h5p', 'h5p', 'h5p']], ['*', 4.0, 'h1p', ['*', 'h2p', 'h2p'],
['*', 'h5p', 'h5p'], 'h6p'], ['*', 4.0, 'h1p', ['*', 'h2p',
'h2p'], 'h5p', ['*', 'h6p', 'h6p']], ['*', 10.0, 'h1p', 'h2p',
```

```
['*', 'h3p', 'h3p', 'h3p'], 'h4p'], ['*', 8.0, 'h1p', 'h2p',
['*', 'h3p', 'h3p', 'h3p'], 'h5p'], ['*', 16.0, 'h1p', 'h2p',
['*', 'h3p', 'h3p', 'h3p'], 'h6p'], ['*', 17.0, 'h1p', 'h2p',
['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p']], ['*', 49.0, 'h1p',
'h2p', ['*', 'h3p', 'h3p'], 'h4p', 'h5p'], ['*', 39.0, 'h1p',
'h2p', ['*', 'h3p', 'h3p'], 'h4p', 'h6p'], ['*', 18.0, 'h1p',
'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p']], ['*', 48.0,
'h1p', 'h2p', ['*', 'h3p', 'h3p'], 'h5p', 'h6p'], ['*', 22.0,
'h1p', 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p']], ['*',
4.0, 'h1p', 'h2p', 'h3p', ['*', 'h4p', 'h4p', 'h4p']], ['*', 23.0,
'h1p', 'h2p', 'h3p', ['*', 'h4p', 'h4p'], 'h5p'], ['*', 13.0,
'h1p', 'h2p', 'h3p', ['*', 'h4p', 'h4p'], 'h6p'], ['*', 23.0,
'h1p', 'h2p', 'h3p', 'h4p', ['*', 'h5p', 'h5p']], ['*', 46.0,
'h1p', 'h2p', 'h3p', 'h4p', 'h5p', 'h6p'], ['*', 14.0, 'h1p',
'h2p', 'h3p', 'h4p', ['*', 'h6p', 'h6p']], ['*', 5.0, 'h1p',
'h2p', 'h3p', ['*', 'h5p', 'h5p', 'h5p']], ['*', 21.0, 'h1p',
'h2p', 'h3p', ['*', 'h5p', 'h5p'], 'h6p'], ['*', 23.0, 'h1p',
'h2p', 'h3p', 'h5p', ['*', 'h6p', 'h6p']], ['*', 5.0, 'h1p',
'h2p', 'h3p', ['*', 'h6p', 'h6p', 'h6p']], ['*', 'h1p', 'h2p',
['*', 'h4p', 'h4p', 'h4p'], 'h5p'], ['*', 5.0, 'h1p', 'h2p', ['*',
'h4p', 'h4p'], ['*', 'h5p', 'h5p']], ['*', 3.0, 'h1p', 'h2p',
['*', 'h4p', 'h4p'], 'h5p', 'h6p'], ['*', 2.0, 'h1p', 'h2p',
'h4p', ['*', 'h5p', 'h5p', 'h5p']], ['*', 9.0, 'h1p', 'h2p',
'h4p', ['*', 'h5p', 'h5p'], 'h6p'], ['*', 3.0, 'h1p', 'h2p',
'h4p', 'h5p', ['*', 'h6p', 'h6p']], ['*', 2.0, 'h1p', 'h2p',
['*', 'h5p', 'h5p', 'h5p'], 'h6p'], ['*', 4.0, 'h1p', 'h2p',
['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*', 'h1p', 'h2p',
'h5p', ['*', 'h6p', 'h6p', 'h6p']], ['*', 12.0, 'h1p', ['*',
'h3p', 'h3p', 'h3p'], ['*', 'h4p', 'h4p']], ['*', 32.0, 'h1p',
['*', 'h3p', 'h3p', 'h3p'], 'h4p', 'h5p'], ['*', 20.0, 'h1p',
['*', 'h3p', 'h3p', 'h3p'], 'h4p', 'h6p'], ['*', 8.0, 'h1p', ['*',
'h3p', 'h3p', 'h3p'], ['*', 'h5p', 'h5p']], ['*', 24.0, 'h1p',
['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'h6p'], ['*', 8.0, 'h1p',
['*', 'h3p', 'h3p', 'h3p'], ['*', 'h6p', 'h6p']], ['*', 6.0,
'h1p', ['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p', 'h4p']], ['*',
26.0, 'h1p', ['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p'], 'h5p'],
['*', 16.0, 'h1p', ['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p'],
'h6p'], ['*', 26.0, 'h1p', ['*', 'h3p', 'h3p'], 'h4p', ['*',
'h5p', 'h5p']], ['*', 50.0, 'h1p', ['*', 'h3p', 'h3p'], 'h4p',
'h5p', 'h6p'], ['*', 14.0, 'h1p', ['*', 'h3p', 'h3p'], 'h4p',
['*', 'h6p', 'h6p']], ['*', 4.0, 'h1p', ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p', 'h5p']], ['*', 24.0, 'h1p', ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], 'h6p'], ['*', 24.0, 'h1p', ['*',
'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p']], ['*', 4.0, 'h1p',
['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p', 'h6p']], ['*', 4.0,
'h1p', 'h3p', ['*', 'h4p', 'h4p', 'h4p'], 'h5p'], ['*', 11.0,
'h1p', 'h3p', ['*', 'h4p', 'h4p'], ['*', 'h5p', 'h5p']], ['*',
12.0, 'h1p', 'h3p', ['*', 'h4p', 'h4p'], 'h5p', 'h6p'], ['*',
3.0, 'h1p', 'h3p', 'h4p', ['*', 'h5p', 'h5p', 'h5p']], ['*',
21.0, 'h1p', 'h3p', 'h4p', ['*', 'h5p', 'h5p'], 'h6p'], ['*',
12.0, 'h1p', 'h3p', 'h4p', 'h5p', ['*', 'h6p', 'h6p']], ['*',
4.0, 'h1p', 'h3p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p'], ['*',
10.0, 'h1p', 'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p']],
['*', 4.0, 'h1p', 'h3p', 'h5p', ['*', 'h6p', 'h6p', 'h6p']],
['*', 'h1p', ['*', 'h4p', 'h4p'], ['*', 'h5p', 'h5p'], 'h6p'],
['*', 'h1p', 'h4p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p'], ['*', 2.0,
'h1p', 'h4p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*',
'h1p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*',
'h1p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p']], ['*',
```

```
['*', 'h2p', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h4p'], ['*',
['*', 'h2p', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h5p'], ['*',
['*', 'h2p', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h6p'], ['*',
['*', 'h2p', 'h2p', 'h2p'], 'h3p', ['*', 'h4p', 'h4p']], ['*',
2.0, ['*', 'h2p', 'h2p', 'h2p'], 'h3p', 'h4p', 'h5p'], ['*', 2.0,
['*', 'h2p', 'h2p', 'h2p'], 'h3p', 'h4p', 'h6p'], ['*', ['*',
'h2p', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p']], ['*', 2.0,
['*', 'h2p', 'h2p', 'h2p'], 'h3p', 'h5p', 'h6p'], ['*', ['*',
'h2p', 'h2p', 'h2p'], 'h3p', ['*', 'h6p', 'h6p']], ['*', ['*',
'h2p', 'h2p', 'h2p'], ['*', 'h4p', 'h4p'], 'h5p'], ['*', ['*',
'h2p', 'h2p', 'h2p'], 'h4p', ['*', 'h5p', 'h5p']], ['*', 2.0,
['*', 'h2p', 'h2p', 'h2p'], 'h4p', 'h5p', 'h6p'], ['*', ['*',
'h2p', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], 'h6p'], ['*', ['*',
'h2p', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p']], ['*', 2.0,
['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h4p'], ['*',
2.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h5p'],
['*', 2.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'],
'h6p'], ['*', 4.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
['*', 'h4p', 'h4p']], ['*', 8.0, ['*', 'h2p', 'h2p'], ['*', 'h3p',
'h3p'], 'h4p', 'h5p'], ['*', 8.0, ['*', 'h2p', 'h2p'], ['*',
'h3p', 'h3p'], 'h4p', 'h6p'], ['*', 4.0, ['*', 'h2p', 'h2p'],
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p']], ['*', 8.0, ['*', 'h2p',
'h2p'], ['*', 'h3p', 'h3p'], 'h5p', 'h6p'], ['*', 4.0, ['*',
'h2p', 'h2p'], ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p']], ['*',
['*', 'h2p', 'h2p'], 'h3p', ['*', 'h4p', 'h4p', 'h4p']], ['*',
7.0, ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h4p', 'h4p'], 'h5p'],
['*', 3.0, ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h4p', 'h4p'],
'h6p'], ['*', 7.0, ['*', 'h2p', 'h2p'], 'h3p', 'h4p', ['*',
'h5p', 'h5p']], ['*', 14.0, ['*', 'h2p', 'h2p'], 'h3p', 'h4p',
'h5p', 'h6p'], ['*', 3.0, ['*', 'h2p', 'h2p'], 'h3p', 'h4p', ['*',
'h6p', 'h6p']], ['*', ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p',
'h5p', 'h5p']], ['*', 7.0, ['*', 'h2p', 'h2p'], 'h3p', ['*',
'h5p', 'h5p'], 'h6p'], ['*', 7.0, ['*', 'h2p', 'h2p'], 'h3p',
'h5p', ['*', 'h6p', 'h6p']], ['*', ['*', 'h2p', 'h2p'], 'h3p',
['*', 'h6p', 'h6p', 'h6p']], ['*', ['*', 'h2p', 'h2p'], ['*',
'h4p', 'h4p', 'h4p'], 'h5p'], ['*', 2.0, ['*', 'h2p', 'h2p'],
['*', 'h4p', 'h4p'], ['*', 'h5p', 'h5p']], ['*', 3.0, ['*',
'h2p', 'h2p'], ['*', 'h4p', 'h4p'], 'h5p', 'h6p'], ['*', ['*',
'h2p', 'h2p'], 'h4p', ['*', 'h5p', 'h5p', 'h5p']], ['*', 4.0,
['*', 'h2p', 'h2p'], 'h4p', ['*', 'h5p', 'h5p'], 'h6p'], ['*',
3.0, ['*', 'h2p', 'h2p'], 'h4p', 'h5p', ['*', 'h6p', 'h6p']],
['*', ['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p', 'h5p'], 'h6p'],
['*', 2.0, ['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p']], ['*', ['*', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p',
'h6p']], ['*', 4.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'], ['*',
'h4p', 'h4p']], ['*', 8.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'],
'h4p', 'h5p'], ['*', 8.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'],
'h4p', 'h6p'], ['*', 4.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'], ['*',
'h5p', 'h5p']], ['*', 8.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'],
'h5p', 'h6p'], ['*', 4.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'],
['*', 'h6p', 'h6p']], ['*', 2.0, 'h2p', ['*', 'h3p', 'h3p'],
['*', 'h4p', 'h4p', 'h4p']], ['*', 14.0, 'h2p', ['*', 'h3p',
'h3p'], ['*', 'h4p', 'h4p'], 'h5p'], ['*', 6.0, 'h2p', ['*',
'h3p', 'h3p'], ['*', 'h4p', 'h4p'], 'h6p'], ['*', 14.0, 'h2p',
['*', 'h3p', 'h3p'], 'h4p', ['*', 'h5p', 'h5p']], ['*', 28.0,
'h2p', ['*', 'h3p', 'h3p'], 'h4p', 'h5p', 'h6p'], ['*', 6.0,
'h2p', ['*', 'h3p', 'h3p'], 'h4p', ['*', 'h6p', 'h6p']], ['*',
2.0, 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p', 'h5p']],
['*', 14.0, 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'],
```

```
'h6p'], ['*', 14.0, 'h2p', ['*', 'h3p', 'h3p'], 'h5p', ['*',
'h6p', 'h6p']], ['*', 2.0, 'h2p', ['*', 'h3p', 'h3p'], ['*',
'h6p', 'h6p', 'h6p']], ['*', 4.0, 'h2p', 'h3p', ['*', 'h4p',
'h4p', 'h4p'], 'h5p'], ['*', 8.0, 'h2p', 'h3p', ['*', 'h4p',
'h4p'], ['*', 'h5p', 'h5p']], ['*', 12.0, 'h2p', 'h3p', ['*',
'h4p', 'h4p'], 'h5p', 'h6p'], ['*', 4.0, 'h2p', 'h3p', 'h4p',
['*', 'h5p', 'h5p', 'h5p']], ['*', 16.0, 'h2p', 'h3p', 'h4p',
['*', 'h5p', 'h5p'], 'h6p'], ['*', 12.0, 'h2p', 'h3p', 'h4p',
'h5p', ['*', 'h6p', 'h6p']], ['*', 4.0, 'h2p', 'h3p', ['*', 'h5p',
'h5p', 'h5p'], 'h6p'], ['*', 8.0, 'h2p', 'h3p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p']], ['*', 4.0, 'h2p', 'h3p', 'h5p',
['*', 'h6p', 'h6p', 'h6p']], ['*', 'h2p', ['*', 'h4p', 'h4p',
'h4p'], ['*', 'h5p', 'h5p']], ['*', 'h2p', ['*', 'h4p', 'h4p'],
['*', 'h5p', 'h5p', 'h5p']], ['*', 3.0, 'h2p', ['*', 'h4p',
'h4p'], ['*', 'h5p', 'h5p'], 'h6p'], ['*', 2.0, 'h2p', 'h4p',
['*', 'h5p', 'h5p', 'h5p'], 'h6p'], ['*', 3.0, 'h2p', 'h4p', ['*',
'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*', 'h2p', ['*', 'h5p',
'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*', 'h2p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p', 'h6p']], ['*', 8.0, ['*', 'h3p',
'h3p', 'h3p'], ['*', 'h4p', 'h4p'], 'h5p'], ['*', 8.0, ['*',
'h3p', 'h3p', 'h3p'], 'h4p', ['*', 'h5p', 'h5p']], ['*', 16.0,
['*', 'h3p', 'h3p', 'h3p'], 'h4p', 'h5p', 'h6p'], ['*', 8.0, ['*',
'h3p', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'h6p'], ['*', 8.0,
['*', 'h3p', 'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p']], ['*',
4.0, ['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p', 'h4p'], 'h5p'],
['*', 8.0, ['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p'], ['*', 'h5p',
'h5p']], ['*', 12.0, ['*', 'h3p', 'h3p'], ['*', 'h4p', 'h4p'],
'h5p', 'h6p'], ['*', 4.0, ['*', 'h3p', 'h3p'], 'h4p', ['*', 'h5p',
'h5p', 'h5p']], ['*', 16.0, ['*', 'h3p', 'h3p'], 'h4p', ['*',
'h5p', 'h5p'], 'h6p'], ['*', 12.0, ['*', 'h3p', 'h3p'], 'h4p',
'h5p', ['*', 'h6p', 'h6p']], ['*', 4.0, ['*', 'h3p', 'h3p'], ['*',
'h5p', 'h5p', 'h5p'], 'h6p'], ['*', 8.0, ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*', 4.0, ['*',
'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p', 'h6p']], ['*', 2.0,
'h3p', ['*', 'h4p', 'h4p', 'h4p'], ['*', 'h5p', 'h5p']], ['*',
2.0, 'h3p', ['*', 'h4p', 'h4p'], ['*', 'h5p', 'h5p', 'h5p']],
['*', 6.0, 'h3p', ['*', 'h4p', 'h4p'], ['*', 'h5p', 'h5p'],
'h6p'], ['*', 4.0, 'h3p', 'h4p', ['*', 'h5p', 'h5p', 'h5p'],
'h6p'], ['*', 6.0, 'h3p', 'h4p', ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p']], ['*', 2.0, 'h3p', ['*', 'h5p', 'h5p', 'h5p'], ['*',
'h6p', 'h6p']], ['*', 2.0, 'h3p', ['*', 'h5p', 'h5p'], ['*',
'h6p', 'h6p', 'h6p']]]],
['+', ['*', -1.0, ['*', 2.0, ['*', 'h1n', 'h1n', 'h1n'], ['*',
'h2n', 'h2n'], 'h3n']], ['*', 2.0, ['*', 'h1n', 'h1n', 'h1n'],
['*', 'h2n', 'h2n'], 'h5n'], ['*', -1.0, ['*', 8.0, ['*', 'h1n',
'h1n', 'h1n'], 'h2n', ['*', 'h3n', 'h3n']]], ['*', 4.0, ['*',
'h1n', 'h1n', 'h1n'], 'h2n', 'h3n', 'h4n'], ['*', 10.0, ['*',
'h1n', 'h1n', 'h1n'], 'h2n', 'h3n', 'h5n'], ['*', 10.0, ['*',
'h1n', 'h1n', 'h1n'], 'h2n', 'h3n', 'h6n'], ['*', 2.0, ['*',
'h1n', 'h1n', 'h1n'], 'h2n', 'h4n', 'h5n'], ['*', 2.0, ['*',
'h1n', 'h1n', 'h1n'], 'h2n', ['*', 'h5n', 'h5n']], ['*', 24.0,
['*', 'h1n', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h4n'], ['*',
16.0, ['*', 'h1n', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h5n'],
['*', 16.0, ['*', 'h1n', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'],
'h6n'], ['*', 6.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n', ['*',
'h4n', 'h4n']], ['*', 6.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n',
'h4n', 'h5n'], ['*', 10.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n',
'h4n', 'h6n'], ['*', 4.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n', ['*',
'h5n', 'h5n']], ['*', 8.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n',
```

```
'h5n', 'h6n'], ['*', 4.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n',
['*', 'h6n', 'h6n']], ['*', 2.0, ['*', 'h1n', 'h1n', 'h1n'],
'h4n', 'h5n', 'h6n'], ['*', 2.0, ['*', 'h1n', 'h1n', 'h1n'],
['*', 'h5n', 'h5n'], 'h6n'], ['*', 2.0, ['*', 'h1n', 'h1n',
'h1n'], 'h5n', ['*', 'h6n', 'h6n']], ['*', -1.0, ['*', ['*',
'h1n', 'h1n'], ['*', 'h2n', 'h2n', 'h2n'], 'h3n']], ['*', ['*',
'h1n', 'h1n'], ['*', 'h2n', 'h2n', 'h2n'], 'h5n'], ['*', -1.0,
['*', 9.0, ['*', 'h1n', 'h1n'], ['*', 'h2n', 'h2n'], ['*', 'h3n',
'h3n']]], ['*', ['*', 'h1n', 'h1n'], ['*', 'h2n', 'h2n'], 'h3n',
'h4n'], ['*', 10.0, ['*', 'h1n', 'h1n'], ['*', 'h2n', 'h2n'],
'h3n', 'h5n'], ['*', 8.0, ['*', 'h1n', 'h1n'], ['*', 'h2n',
'h2n'], 'h3n', 'h6n'], ['*', 6.0, ['*', 'h1n', 'h1n'], ['*',
'h2n', 'h2n'], 'h4n', 'h5n'], ['*', 3.0, ['*', 'h1n', 'h1n'],
['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n']], ['*', 3.0, ['*', 'h1n',
'h1n'], ['*', 'h2n', 'h2n'], 'h5n', 'h6n'], ['*', -1.0, ['*',
8.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h3n', 'h3n', 'h3n']]],
['*', 28.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h3n', 'h3n'],
'h4n'], ['*', 26.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h3n',
'h3n'], 'h5n'], ['*', 34.0, ['*', 'h1n', 'h1n'], 'h2n', ['*',
'h3n', 'h3n'], 'h6n'], ['*', 13.0, ['*', 'h1n', 'h1n'], 'h2n',
'h3n', ['*', 'h4n', 'h4n']], ['*', 28.0, ['*', 'h1n', 'h1n'],
'h2n', 'h3n', 'h4n', 'h5n'], ['*', 30.0, ['*', 'h1n', 'h1n'],
'h2n', 'h3n', 'h4n', 'h6n'], ['*', 15.0, ['*', 'h1n', 'h1n'],
'h2n', 'h3n', ['*', 'h5n', 'h5n']], ['*', 32.0, ['*', 'h1n',
'h1n'], 'h2n', 'h3n', 'h5n', 'h6n'], ['*', 17.0, ['*', 'h1n',
'h1n'], 'h2n', 'h3n', ['*', 'h6n', 'h6n']], ['*', 3.0, ['*',
'h1n', 'h1n'], 'h2n', ['*', 'h4n', 'h4n'], 'h5n'], ['*', 6.0,
['*', 'h1n', 'h1n'], 'h2n', 'h4n', ['*', 'h5n', 'h5n']], ['*',
6.0, ['*', 'h1n', 'h1n'], 'h2n', 'h4n', 'h5n', 'h6n'], ['*', ['*',
'h1n', 'h1n'], 'h2n', ['*', 'h5n', 'h5n', 'h5n']], ['*', 4.0,
['*', 'h1n', 'h1n'], 'h2n', ['*', 'h5n', 'h5n'], 'h6n'], ['*',
3.0, ['*', 'h1n', 'h1n'], 'h2n', 'h5n', ['*', 'h6n', 'h6n']],
['*', 24.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n', 'h3n'],
'h4n'], ['*', 16.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n',
'h3n'], 'h5n'], ['*', 16.0, ['*', 'h1n', 'h1n'], ['*', 'h3n',
'h3n', 'h3n'], 'h6n'], ['*', 21.0, ['*', 'h1n', 'h1n'], ['*',
'h3n', 'h3n'], ['*', 'h4n', 'h4n']], ['*', 46.0, ['*', 'h1n',
'h1n'], ['*', 'h3n', 'h3n'], 'h4n', 'h5n'], ['*', 38.0, ['*',
'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h4n', 'h6n'], ['*', 16.0,
['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n']],
['*', 40.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h5n',
'h6n'], ['*', 16.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'],
['*', 'h6n', 'h6n']], ['*', 3.0, ['*', 'h1n', 'h1n'], 'h3n', ['*',
'h4n', 'h4n', 'h4n']], ['*', 14.0, ['*', 'h1n', 'h1n'], 'h3n',
['*', 'h4n', 'h4n'], 'h5n'], ['*', 8.0, ['*', 'h1n', 'h1n'],
'h3n', ['*', 'h4n', 'h4n'], 'h6n'], ['*', 11.0, ['*', 'h1n',
'h1n'], 'h3n', 'h4n', ['*', 'h5n', 'h5n']], ['*', 24.0, ['*',
'h1n', 'h1n'], 'h3n', 'h4n', 'h5n', 'h6n'], ['*', 7.0, ['*',
'h1n', 'h1n'], 'h3n', 'h4n', ['*', 'h6n', 'h6n']], ['*', 2.0,
['*', 'h1n', 'h1n'], 'h3n', ['*', 'h5n', 'h5n', 'h5n']], ['*',
12.0, ['*', 'h1n', 'h1n'], 'h3n', ['*', 'h5n', 'h5n'], 'h6n'],
['*', 12.0, ['*', 'h1n', 'h1n'], 'h3n', 'h5n', ['*', 'h6n',
'h6n']], ['*', 2.0, ['*', 'h1n', 'h1n'], 'h3n', ['*', 'h6n',
'h6n', 'h6n']], ['*', ['*', 'h1n', 'h1n'], ['*', 'h4n', 'h4n'],
'h5n', 'h6n'], ['*', 4.0, ['*', 'h1n', 'h1n'], 'h4n', ['*', 'h5n',
'h5n'], 'h6n'], ['*', 2.0, ['*', 'h1n', 'h1n'], 'h4n', 'h5n',
['*', 'h6n', 'h6n']], ['*', ['*', 'h1n', 'h1n'], ['*', 'h5n',
'h5n', 'h5n'], 'h6n'], ['*', 3.0, ['*', 'h1n', 'h1n'], ['*',
'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*', ['*', 'h1n', 'h1n'],
```

```
'h5n', ['*', 'h6n', 'h6n', 'h6n']], ['*', -1.0, ['*', 'h1n',
['*', 'h2n', 'h2n', 'h2n'], ['*', 'h3n', 'h3n']]], ['*', 'h1n',
['*', 'h2n', 'h2n', 'h2n'], 'h3n', 'h5n'], ['*', 2.0, 'h1n',
['*', 'h2n', 'h2n', 'h2n'], 'h3n', 'h6n'], ['*', 2.0, 'h1n',
['*', 'h2n', 'h2n', 'h2n'], 'h4n', 'h5n'], ['*', 'h1n', ['*',
'h2n', 'h2n', 'h2n'], ['*', 'h5n', 'h5n']], ['*', 'h1n', ['*',
'h2n', 'h2n', 'h2n'], 'h5n', 'h6n'], ['*', -1.0, ['*', 2.0, 'h1n',
['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n']]], ['*', 8.0,
'h1n', ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h4n'], ['*',
9.0, 'h1n', ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h5n'],
['*', 15.0, 'h1n', ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
'h6n'], ['*', 6.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h4n',
'h4n']], ['*', 20.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', 'h4n',
'h5n'], ['*', 15.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', 'h4n',
'h6n'], ['*', 8.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n',
'h5n']], ['*', 19.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', 'h5n',
'h6n'], ['*', 9.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h6n',
'h6n']], ['*', 5.0, 'h1n', ['*', 'h2n', 'h2n'], ['*', 'h4n',
'h4n'], 'h5n'], ['*', 5.0, 'h1n', ['*', 'h2n', 'h2n'], 'h4n',
['*', 'h5n', 'h5n']], ['*', 9.0, 'h1n', ['*', 'h2n', 'h2n'],
'h4n', 'h5n', 'h6n'], ['*', 'h1n', ['*', 'h2n', 'h2n'], ['*',
'h5n', 'h5n', 'h5n']], ['*', 4.0, 'h1n', ['*', 'h2n', 'h2n'],
['*', 'h5n', 'h5n'], 'h6n'], ['*', 4.0, 'h1n', ['*', 'h2n',
'h2n'], 'h5n', ['*', 'h6n', 'h6n']], ['*', 10.0, 'h1n', 'h2n',
['*', 'h3n', 'h3n', 'h3n'], 'h4n'], ['*', 8.0, 'h1n', 'h2n',
['*', 'h3n', 'h3n', 'h3n'], 'h5n'], ['*', 16.0, 'h1n', 'h2n',
['*', 'h3n', 'h3n', 'h3n'], 'h6n'], ['*', 17.0, 'h1n', 'h2n',
['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n']], ['*', 49.0, 'h1n',
'h2n', ['*', 'h3n', 'h3n'], 'h4n', 'h5n'], ['*', 39.0, 'h1n',
'h2n', ['*', 'h3n', 'h3n'], 'h4n', 'h6n'], ['*', 18.0, 'h1n',
'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n']], ['*', 48.0,
'h1n', 'h2n', ['*', 'h3n', 'h3n'], 'h5n', 'h6n'], ['*', 22.0,
'h1n', 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n']], ['*',
4.0, 'h1n', 'h2n', 'h3n', ['*', 'h4n', 'h4n', 'h4n']], ['*', 23.0,
'h1n', 'h2n', 'h3n', ['*', 'h4n', 'h4n'], 'h5n'], ['*', 13.0,
'h1n', 'h2n', 'h3n', ['*', 'h4n', 'h4n'], 'h6n'], ['*', 23.0,
'h1n', 'h2n', 'h3n', 'h4n', ['*', 'h5n', 'h5n']], ['*', 46.0,
'h1n', 'h2n', 'h3n', 'h4n', 'h5n', 'h6n'], ['*', 14.0, 'h1n',
'h2n', 'h3n', 'h4n', ['*', 'h6n', 'h6n']], ['*', 5.0, 'h1n',
'h2n', 'h3n', ['*', 'h5n', 'h5n', 'h5n']], ['*', 21.0, 'h1n',
'h2n', 'h3n', ['*', 'h5n', 'h5n'], 'h6n'], ['*', 23.0, 'h1n',
'h2n', 'h3n', 'h5n', ['*', 'h6n', 'h6n']], ['*', 5.0, 'h1n',
'h2n', 'h3n', ['*', 'h6n', 'h6n', 'h6n']], ['*', 'h1n', 'h2n',
['*', 'h4n', 'h4n', 'h4n'], 'h5n'], ['*', 5.0, 'h1n', 'h2n', ['*',
'h4n', 'h4n'], ['*', 'h5n', 'h5n']], ['*', 3.0, 'h1n', 'h2n',
['*', 'h4n', 'h4n'], 'h5n', 'h6n'], ['*', 2.0, 'h1n', 'h2n',
'h4n', ['*', 'h5n', 'h5n', 'h5n']], ['*', 9.0, 'h1n', 'h2n',
'h4n', ['*', 'h5n', 'h5n'], 'h6n'], ['*', 3.0, 'h1n', 'h2n',
'h4n', 'h5n', ['*', 'h6n', 'h6n']], ['*', 2.0, 'h1n', 'h2n',
['*', 'h5n', 'h5n', 'h5n'], 'h6n'], ['*', 4.0, 'h1n', 'h2n',
['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*', 'h1n', 'h2n',
'h5n', ['*', 'h6n', 'h6n', 'h6n']], ['*', 12.0, 'h1n', ['*',
'h3n', 'h3n', 'h3n'], ['*', 'h4n', 'h4n']], ['*', 32.0, 'h1n',
['*', 'h3n', 'h3n', 'h3n'], 'h4n', 'h5n'], ['*', 20.0, 'h1n',
['*', 'h3n', 'h3n', 'h3n'], 'h4n', 'h6n'], ['*', 8.0, 'h1n', ['*',
'h3n', 'h3n', 'h3n'], ['*', 'h5n', 'h5n']], ['*', 24.0, 'h1n',
['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'h6n'], ['*', 8.0, 'h1n',
['*', 'h3n', 'h3n', 'h3n'], ['*', 'h6n', 'h6n']], ['*', 6.0,
'h1n', ['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n', 'h4n']], ['*',
```

```
26.0, 'h1n', ['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n'], 'h5n'],
['*', 16.0, 'h1n', ['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n'],
'h6n'], ['*', 26.0, 'h1n', ['*', 'h3n', 'h3n'], 'h4n', ['*',
'h5n', 'h5n']], ['*', 50.0, 'h1n', ['*', 'h3n', 'h3n'], 'h4n',
'h5n', 'h6n'], ['*', 14.0, 'h1n', ['*', 'h3n', 'h3n'], 'h4n',
['*', 'h6n', 'h6n']], ['*', 4.0, 'h1n', ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n', 'h5n']], ['*', 24.0, 'h1n', ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], 'h6n'], ['*', 24.0, 'h1n', ['*',
'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n']], ['*', 4.0, 'h1n',
['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n', 'h6n']], ['*', 4.0,
'h1n', 'h3n', ['*', 'h4n', 'h4n', 'h4n'], 'h5n'], ['*', 11.0,
'h1n', 'h3n', ['*', 'h4n', 'h4n'], ['*', 'h5n', 'h5n']], ['*',
12.0, 'h1n', 'h3n', ['*', 'h4n', 'h4n'], 'h5n', 'h6n'], ['*',
3.0, 'h1n', 'h3n', 'h4n', ['*', 'h5n', 'h5n', 'h5n']], ['*',
21.0, 'h1n', 'h3n', 'h4n', ['*', 'h5n', 'h5n'], 'h6n'], ['*',
12.0, 'h1n', 'h3n', 'h4n', 'h5n', ['*', 'h6n', 'h6n']], ['*',
4.0, 'h1n', 'h3n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n'], ['*',
10.0, 'h1n', 'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n']],
['*', 4.0, 'h1n', 'h3n', 'h5n', ['*', 'h6n', 'h6n', 'h6n']],
['*', 'h1n', ['*', 'h4n', 'h4n'], ['*', 'h5n', 'h5n'], 'h6n'],
['*', 'h1n', 'h4n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n'], ['*', 2.0,
'h1n', 'h4n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*',
'h1n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*',
'h1n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n']], ['*',
['*', 'h2n', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h4n'], ['*',
['*', 'h2n', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h5n'], ['*',
['*', 'h2n', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h6n'], ['*',
['*', 'h2n', 'h2n', 'h2n'], 'h3n', ['*', 'h4n', 'h4n']], ['*',
2.0, ['*', 'h2n', 'h2n', 'h2n'], 'h3n', 'h4n', 'h5n'], ['*', 2.0,
['*', 'h2n', 'h2n', 'h2n'], 'h3n', 'h4n', 'h6n'], ['*', ['*',
'h2n', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n']], ['*', 2.0,
['*', 'h2n', 'h2n', 'h2n'], 'h3n', 'h5n', 'h6n'], ['*', ['*',
'h2n', 'h2n', 'h2n'], 'h3n', ['*', 'h6n', 'h6n']], ['*', ['*',
'h2n', 'h2n', 'h2n'], ['*', 'h4n', 'h4n'], 'h5n'], ['*', ['*',
'h2n', 'h2n', 'h2n'], 'h4n', ['*', 'h5n', 'h5n']], ['*', 2.0,
['*', 'h2n', 'h2n', 'h2n'], 'h4n', 'h5n', 'h6n'], ['*', ['*',
'h2n', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], 'h6n'], ['*', ['*',
'h2n', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n']], ['*', 2.0,
['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h4n'], ['*',
2.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h5n'],
['*', 2.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'],
'h6n'], ['*', 4.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
['*', 'h4n', 'h4n']], ['*', 8.0, ['*', 'h2n', 'h2n'], ['*', 'h3n',
'h3n'], 'h4n', 'h5n'], ['*', 8.0, ['*', 'h2n', 'h2n'], ['*',
'h3n', 'h3n'], 'h4n', 'h6n'], ['*', 4.0, ['*', 'h2n', 'h2n'],
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n']], ['*', 8.0, ['*', 'h2n',
'h2n'], ['*', 'h3n', 'h3n'], 'h5n', 'h6n'], ['*', 4.0, ['*',
'h2n', 'h2n'], ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n']], ['*',
['*', 'h2n', 'h2n'], 'h3n', ['*', 'h4n', 'h4n', 'h4n']], ['*',
7.0, ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h4n', 'h4n'], 'h5n'],
['*', 3.0, ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h4n', 'h4n'],
'h6n'], ['*', 7.0, ['*', 'h2n', 'h2n'], 'h3n', 'h4n', ['*',
'h5n', 'h5n']], ['*', 14.0, ['*', 'h2n', 'h2n'], 'h3n', 'h4n',
'h5n', 'h6n'], ['*', 3.0, ['*', 'h2n', 'h2n'], 'h3n', 'h4n', ['*',
'h6n', 'h6n']], ['*', ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n',
'h5n', 'h5n']], ['*', 7.0, ['*', 'h2n', 'h2n'], 'h3n', ['*',
'h5n', 'h5n'], 'h6n'], ['*', 7.0, ['*', 'h2n', 'h2n'], 'h3n',
'h5n', ['*', 'h6n', 'h6n']], ['*', ['*', 'h2n', 'h2n'], 'h3n',
['*', 'h6n', 'h6n', 'h6n']], ['*', ['*', 'h2n', 'h2n'], ['*',
```

```
'h4n', 'h4n', 'h4n'], 'h5n'], ['*', 2.0, ['*', 'h2n', 'h2n'],
['*', 'h4n', 'h4n'], ['*', 'h5n', 'h5n']], ['*', 3.0, ['*',
'h2n', 'h2n'], ['*', 'h4n', 'h4n'], 'h5n', 'h6n'], ['*', ['*',
'h2n', 'h2n'], 'h4n', ['*', 'h5n', 'h5n', 'h5n']], ['*', 4.0,
['*', 'h2n', 'h2n'], 'h4n', ['*', 'h5n', 'h5n'], 'h6n'], ['*',
3.0, ['*', 'h2n', 'h2n'], 'h4n', 'h5n', ['*', 'h6n', 'h6n']],
['*', ['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n', 'h5n'], 'h6n'],
['*', 2.0, ['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n']], ['*', ['*', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n',
'h6n']], ['*', 4.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'], ['*',
'h4n', 'h4n']], ['*', 8.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'],
'h4n', 'h5n'], ['*', 8.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'],
'h4n', 'h6n'], ['*', 4.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'], ['*',
'h5n', 'h5n']], ['*', 8.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'],
'h5n', 'h6n'], ['*', 4.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'],
['*', 'h6n', 'h6n']], ['*', 2.0, 'h2n', ['*', 'h3n', 'h3n'],
['*', 'h4n', 'h4n', 'h4n']], ['*', 14.0, 'h2n', ['*', 'h3n',
'h3n'], ['*', 'h4n', 'h4n'], 'h5n'], ['*', 6.0, 'h2n', ['*',
'h3n', 'h3n'], ['*', 'h4n', 'h4n'], 'h6n'], ['*', 14.0, 'h2n',
['*', 'h3n', 'h3n'], 'h4n', ['*', 'h5n', 'h5n']], ['*', 28.0,
'h2n', ['*', 'h3n', 'h3n'], 'h4n', 'h5n', 'h6n'], ['*', 6.0,
'h2n', ['*', 'h3n', 'h3n'], 'h4n', ['*', 'h6n', 'h6n']], ['*',
2.0, 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n', 'h5n']],
['*', 14.0, 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'],
'h6n'], ['*', 14.0, 'h2n', ['*', 'h3n', 'h3n'], 'h5n', ['*',
'h6n', 'h6n']], ['*', 2.0, 'h2n', ['*', 'h3n', 'h3n'], ['*',
'h6n', 'h6n', 'h6n']], ['*', 4.0, 'h2n', 'h3n', ['*', 'h4n',
'h4n', 'h4n'], 'h5n'], ['*', 8.0, 'h2n', 'h3n', ['*', 'h4n',
'h4n'], ['*', 'h5n', 'h5n']], ['*', 12.0, 'h2n', 'h3n', ['*',
'h4n', 'h4n'], 'h5n', 'h6n'], ['*', 4.0, 'h2n', 'h3n', 'h4n',
['*', 'h5n', 'h5n', 'h5n']], ['*', 16.0, 'h2n', 'h3n', 'h4n',
['*', 'h5n', 'h5n'], 'h6n'], ['*', 12.0, 'h2n', 'h3n', 'h4n',
'h5n', ['*', 'h6n', 'h6n']], ['*', 4.0, 'h2n', 'h3n', ['*', 'h5n',
'h5n', 'h5n'], 'h6n'], ['*', 8.0, 'h2n', 'h3n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n']], ['*', 4.0, 'h2n', 'h3n', 'h5n',
['*', 'h6n', 'h6n', 'h6n']], ['*', 'h2n', ['*', 'h4n', 'h4n',
'h4n'], ['*', 'h5n', 'h5n']], ['*', 'h2n', ['*', 'h4n', 'h4n'],
['*', 'h5n', 'h5n', 'h5n']], ['*', 3.0, 'h2n', ['*', 'h4n',
'h4n'], ['*', 'h5n', 'h5n'], 'h6n'], ['*', 2.0, 'h2n', 'h4n',
['*', 'h5n', 'h5n', 'h5n'], 'h6n'], ['*', 3.0, 'h2n', 'h4n', ['*',
'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*', 'h2n', ['*', 'h5n',
'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*', 'h2n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n', 'h6n']], ['*', 8.0, ['*', 'h3n',
'h3n', 'h3n'], ['*', 'h4n', 'h4n'], 'h5n'], ['*', 8.0, ['*',
'h3n', 'h3n', 'h3n'], 'h4n', ['*', 'h5n', 'h5n']], ['*', 16.0,
['*', 'h3n', 'h3n', 'h3n'], 'h4n', 'h5n', 'h6n'], ['*', 8.0, ['*',
'h3n', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'h6n'], ['*', 8.0,
['*', 'h3n', 'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n']], ['*',
4.0, ['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n', 'h4n'], 'h5n'],
['*', 8.0, ['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n'], ['*', 'h5n',
'h5n']], ['*', 12.0, ['*', 'h3n', 'h3n'], ['*', 'h4n', 'h4n'],
'h5n', 'h6n'], ['*', 4.0, ['*', 'h3n', 'h3n'], 'h4n', ['*', 'h5n',
'h5n', 'h5n']], ['*', 16.0, ['*', 'h3n', 'h3n'], 'h4n', ['*',
'h5n', 'h5n'], 'h6n'], ['*', 12.0, ['*', 'h3n', 'h3n'], 'h4n',
'h5n', ['*', 'h6n', 'h6n']], ['*', 4.0, ['*', 'h3n', 'h3n'], ['*',
'h5n', 'h5n', 'h5n'], 'h6n'], ['*', 8.0, ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*', 4.0, ['*',
'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n', 'h6n']], ['*', 2.0,
'h3n', ['*', 'h4n', 'h4n', 'h4n'], ['*', 'h5n', 'h5n']], ['*',
```

```
2.0, 'h3n', ['*', 'h4n', 'h4n'], ['*', 'h5n', 'h5n', 'h5n']],
['*', 6.0, 'h3n', ['*', 'h4n', 'h4n'], ['*', 'h5n', 'h5n'],
'h6n'], ['*', 4.0, 'h3n', 'h4n', ['*', 'h5n', 'h5n', 'h5n'],
'h6n'], ['*', 6.0, 'h3n', 'h4n', ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n']], ['*', 2.0, 'h3n', ['*', 'h5n', 'h5n', 'h5n'], ['*',
'h6n', 'h6n']], ['*', 2.0, 'h3n', ['*', 'h5n', 'h5n'], ['*',
'h6n', 'h6n', 'h6n']]]
]

Another example:
[
['-', 0.0, ['+', ['*', ['*', 'h1p', 'h1p', 'h1p'], ['*', 'h3p',
'h3p'], 'h5p', ['*', 'j2p', 'j2p']], ['*', ['*', 'h1p', 'h1p',
'h1p'], ['*', 'h3p', 'h3p'], 'h5p', 'j2p'], ['*', 4.0, ['*',
'h1p', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h6p', ['*', 'j2p',
'j2p']], ['*', 4.0, ['*', 'h1p', 'h1p', 'h1p'], ['*', 'h3p',
'h3p'], 'h6p', 'j2p'], ['*', ['*', 'h1p', 'h1p', 'h1p'], 'h3p',
['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*', ['*', 'h1p',
'h1p', 'h1p'], 'h3p', 'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 5.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p', 'h5p', 'h6p', ['*',
'j2p', 'j2p']], ['*', 2.0, ['*', 'h1p', 'h1p', 'h1p'], 'h3p',
'h5p', 'h6p', 'j2p'], ['*', 4.0, ['*', 'h1p', 'h1p', 'h1p'],
'h3p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 4.0,
['*', 'h1p', 'h1p', 'h1p'], 'h3p', ['*', 'h6p', 'h6p'], 'j2p'],
['*', ['*', 'h1p', 'h1p', 'h1p'], ['*', 'h5p', 'h5p'], 'h6p',
['*', 'j2p', 'j2p', 'j2p']], ['*', ['*', 'h1p', 'h1p', 'h1p'],
['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', ['*',
'h1p', 'h1p', 'h1p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p']], ['*', 2.0, ['*', 'h1p', 'h1p', 'h1p'], 'h5p',
['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', ['*', 'h1p',
'h1p', 'h1p'], 'h5p', ['*', 'h6p', 'h6p'], 'j2p'], ['*', 5.0,
['*', 'h1p', 'h1p'], 'h2p', ['*', 'h3p', 'h3p'], 'h5p', ['*',
'j2p', 'j2p']], ['*', 6.0, ['*', 'h1p', 'h1p'], 'h2p', ['*',
'h3p', 'h3p'], 'h5p', 'j2p'], ['*', ['*', 'h1p', 'h1p'], 'h2p',
['*', 'h3p', 'h3p'], 'h5p'], ['*', 12.0, ['*', 'h1p', 'h1p'],
'h2p', ['*', 'h3p', 'h3p'], 'h6p', ['*', 'j2p', 'j2p']], ['*',
12.0, ['*', 'h1p', 'h1p'], 'h2p', ['*', 'h3p', 'h3p'], 'h6p',
'j2p'], ['*', 5.0, ['*', 'h1p', 'h1p'], 'h2p', 'h3p', ['*',
'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*', 2.0, ['*', 'h1p',
'h1p'], 'h2p', 'h3p', ['*', 'h5p', 'h5p'], 'j2p'], ['*', 3.0,
['*', 'h1p', 'h1p'], 'h2p', 'h3p', 'h5p', 'h6p', ['*', 'j2p',
'j2p', 'j2p']], ['*', 16.0, ['*', 'h1p', 'h1p'], 'h2p', 'h3p',
'h5p', 'h6p', ['*', 'j2p', 'j2p']], ['*', 11.0, ['*', 'h1p',
'h1p'], 'h2p', 'h3p', 'h5p', 'h6p', 'j2p'], ['*', 2.0, ['*',
'h1p', 'h1p'], 'h2p', 'h3p', 'h5p', 'h6p'], ['*', 12.0, ['*',
'h1p', 'h1p'], 'h2p', 'h3p', ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 12.0, ['*', 'h1p', 'h1p'], 'h2p', 'h3p', ['*',
'h6p', 'h6p'], 'j2p'], ['*', 3.0, ['*', 'h1p', 'h1p'], 'h2p',
['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*',
5.0, ['*', 'h1p', 'h1p'], 'h2p', ['*', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p']], ['*', 2.0, ['*', 'h1p', 'h1p'], 'h2p', ['*',
'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 3.0, ['*', 'h1p', 'h1p'],
'h2p', 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']],
['*', 7.0, ['*', 'h1p', 'h1p'], 'h2p', 'h5p', ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 5.0, ['*', 'h1p', 'h1p'], 'h2p',
'h5p', ['*', 'h6p', 'h6p'], 'j2p'], ['*', ['*', 'h1p', 'h1p'],
'h2p', 'h5p', ['*', 'h6p', 'h6p']], ['*', ['*', 'h1p', 'h1p'],
['*', 'h3p', 'h3p', 'h3p'], 'h5p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 2.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p', 'h3p'], 'h5p',
```

```
['*', 'j2p', 'j2p']], ['*', ['*', 'h1p', 'h1p'], ['*', 'h3p',
'h3p', 'h3p'], 'h5p', 'j2p'], ['*', 4.0, ['*', 'h1p', 'h1p'],
['*', 'h3p', 'h3p', 'h3p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 8.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p', 'h3p'], 'h6p',
['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h1p', 'h1p'], ['*', 'h3p',
'h3p', 'h3p'], 'h6p', 'j2p'], ['*', 2.0, ['*', 'h1p', 'h1p'],
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 4.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*', 2.0, ['*', 'h1p',
'h1p'], ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'j2p'], ['*',
['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h5p', 'h6p', ['*',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 12.0, ['*', 'h1p', 'h1p'],
['*', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 25.0, ['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], 'h5p',
'h6p', ['*', 'j2p', 'j2p']], ['*', 14.0, ['*', 'h1p', 'h1p'],
['*', 'h3p', 'h3p'], 'h5p', 'h6p', 'j2p'], ['*', 8.0, ['*', 'h1p',
'h1p'], ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p']], ['*', 16.0, ['*', 'h1p', 'h1p'], ['*', 'h3p',
'h3p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 8.0,
['*', 'h1p', 'h1p'], ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'],
'j2p'], ['*', ['*', 'h1p', 'h1p'], 'h3p', ['*', 'h5p', 'h5p',
'h5p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', ['*', 'h1p', 'h1p'],
'h3p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*',
2.0, ['*', 'h1p', 'h1p'], 'h3p', ['*', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 11.0, ['*', 'h1p', 'h1p'],
'h3p', ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 13.0, ['*', 'h1p', 'h1p'], 'h3p', ['*', 'h5p', 'h5p'],
'h6p', ['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h1p', 'h1p'],
'h3p', ['*', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 2.0, ['*', 'h1p',
'h1p'], 'h3p', 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p', 'j2p']], ['*', 14.0, ['*', 'h1p', 'h1p'], 'h3p', 'h5p',
['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 26.0,
['*', 'h1p', 'h1p'], 'h3p', 'h5p', ['*', 'h6p', 'h6p'], ['*',
'j2p', 'j2p']], ['*', 14.0, ['*', 'h1p', 'h1p'], 'h3p', 'h5p',
['*', 'h6p', 'h6p'], 'j2p'], ['*', 4.0, ['*', 'h1p', 'h1p'],
'h3p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']],
['*', 8.0, ['*', 'h1p', 'h1p'], 'h3p', ['*', 'h6p', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h1p', 'h1p'], 'h3p', ['*',
'h6p', 'h6p', 'h6p'], 'j2p'], ['*', ['*', 'h1p', 'h1p'], ['*',
'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p', 'j2p']],
['*', 2.0, ['*', 'h1p', 'h1p'], ['*', 'h5p', 'h5p', 'h5p'], 'h6p',
['*', 'j2p', 'j2p', 'j2p']], ['*', ['*', 'h1p', 'h1p'], ['*',
'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 2.0,
['*', 'h1p', 'h1p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 6.0, ['*', 'h1p',
'h1p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p']], ['*', 6.0, ['*', 'h1p', 'h1p'], ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 2.0,
['*', 'h1p', 'h1p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
'j2p'], ['*', ['*', 'h1p', 'h1p'], 'h5p', ['*', 'h6p', 'h6p',
'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 3.0, ['*',
'h1p', 'h1p'], 'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p']], ['*', 3.0, ['*', 'h1p', 'h1p'], 'h5p', ['*',
'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', ['*', 'h1p',
'h1p'], 'h5p', ['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*', 7.0,
'h1p', ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h5p', ['*',
'j2p', 'j2p']], ['*', 9.0, 'h1p', ['*', 'h2p', 'h2p'], ['*',
'h3p', 'h3p'], 'h5p', 'j2p'], ['*', 2.0, 'h1p', ['*', 'h2p',
'h2p'], ['*', 'h3p', 'h3p'], 'h5p'], ['*', 12.0, 'h1p', ['*',
```

```
'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h6p', ['*', 'j2p', 'j2p']],
['*', 12.0, 'h1p', ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
'h6p', 'j2p'], ['*', 7.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p',
['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*', 6.0, 'h1p',
['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p'], 'j2p'], ['*',
'h1p', ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p']], ['*',
3.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', 'h5p', 'h6p', ['*', 'j2p',
'j2p', 'j2p']], ['*', 17.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p',
'h5p', 'h6p', ['*', 'j2p', 'j2p']], ['*', 16.0, 'h1p', ['*',
'h2p', 'h2p'], 'h3p', 'h5p', 'h6p', 'j2p'], ['*', 4.0, 'h1p',
['*', 'h2p', 'h2p'], 'h3p', 'h5p', 'h6p'], ['*', 12.0, 'h1p',
['*', 'h2p', 'h2p'], 'h3p', ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 12.0, 'h1p', ['*', 'h2p', 'h2p'], 'h3p', ['*',
'h6p', 'h6p'], 'j2p'], ['*', 3.0, 'h1p', ['*', 'h2p', 'h2p'],
['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*',
7.0, 'h1p', ['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], 'h6p',
['*', 'j2p', 'j2p']], ['*', 5.0, 'h1p', ['*', 'h2p', 'h2p'],
['*', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 'h1p', ['*', 'h2p',
'h2p'], ['*', 'h5p', 'h5p'], 'h6p'], ['*', 3.0, 'h1p', ['*',
'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 8.0, 'h1p', ['*', 'h2p', 'h2p'], 'h5p', ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 7.0, 'h1p', ['*',
'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p'], 'j2p'], ['*', 2.0,
'h1p', ['*', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p']], ['*',
4.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p', 'h3p'], 'h5p', ['*', 'j2p',
'j2p', 'j2p']], ['*', 9.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p',
'h3p'], 'h5p', ['*', 'j2p', 'j2p']], ['*', 6.0, 'h1p', 'h2p',
['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'j2p'], ['*', 'h1p', 'h2p',
['*', 'h3p', 'h3p', 'h3p'], 'h5p'], ['*', 8.0, 'h1p', 'h2p', ['*',
'h3p', 'h3p', 'h3p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*',
16.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p', 'h3p'], 'h6p', ['*',
'j2p', 'j2p']], ['*', 8.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p',
'h3p'], 'h6p', 'j2p'], ['*', 8.0, 'h1p', 'h2p', ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
17.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'],
['*', 'j2p', 'j2p']], ['*', 12.0, 'h1p', 'h2p', ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], 'j2p'], ['*', 2.0, 'h1p', 'h2p',
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p']], ['*', 2.0, 'h1p',
'h2p', ['*', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p',
'j2p', 'j2p']], ['*', 25.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p'],
'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*', 47.0, 'h1p',
'h2p', ['*', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p']],
['*', 31.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p'], 'h5p', 'h6p',
'j2p'], ['*', 3.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p'], 'h5p',
'h6p'], ['*', 16.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p'], ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 32.0, 'h1p',
'h2p', ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 16.0, 'h1p', 'h2p', ['*', 'h3p', 'h3p'], ['*',
'h6p', 'h6p'], 'j2p'], ['*', 4.0, 'h1p', 'h2p', 'h3p', ['*',
'h5p', 'h5p', 'h5p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 6.0,
'h1p', 'h2p', 'h3p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'j2p',
'j2p']], ['*', 2.0, 'h1p', 'h2p', 'h3p', ['*', 'h5p', 'h5p',
'h5p'], 'j2p'], ['*', 4.0, 'h1p', 'h2p', 'h3p', ['*', 'h5p',
'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 23.0,
'h1p', 'h2p', 'h3p', ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p',
'j2p', 'j2p']], ['*', 36.0, 'h1p', 'h2p', 'h3p', ['*', 'h5p',
'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 21.0, 'h1p', 'h2p',
'h3p', ['*', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 4.0, 'h1p',
'h2p', 'h3p', ['*', 'h5p', 'h5p'], 'h6p'], ['*', 4.0, 'h1p',
```

```
'h2p', 'h3p', 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p', 'j2p']], ['*', 28.0, 'h1p', 'h2p', 'h3p', 'h5p', ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 51.0, 'h1p',
'h2p', 'h3p', 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']],
['*', 30.0, 'h1p', 'h2p', 'h3p', 'h5p', ['*', 'h6p', 'h6p'],
'j2p'], ['*', 3.0, 'h1p', 'h2p', 'h3p', 'h5p', ['*', 'h6p',
'h6p']], ['*', 8.0, 'h1p', 'h2p', 'h3p', ['*', 'h6p', 'h6p',
'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 16.0, 'h1p', 'h2p',
'h3p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*',
8.0, 'h1p', 'h2p', 'h3p', ['*', 'h6p', 'h6p', 'h6p'], 'j2p'],
['*', 2.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 6.0, 'h1p', 'h2p', ['*',
'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*',
6.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p',
'j2p']], ['*', 2.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p', 'h5p'],
'h6p', 'j2p'], ['*', 4.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p'],
['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*',
14.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p']], ['*', 18.0, 'h1p', 'h2p', ['*',
'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*',
10.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
'j2p'], ['*', 2.0, 'h1p', 'h2p', ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p']], ['*', 2.0, 'h1p', 'h2p', 'h5p', ['*', 'h6p', 'h6p',
'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 7.0, 'h1p',
'h2p', 'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 9.0, 'h1p', 'h2p', 'h5p', ['*', 'h6p', 'h6p',
'h6p'], ['*', 'j2p', 'j2p']], ['*', 5.0, 'h1p', 'h2p', 'h5p',
['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*', 'h1p', 'h2p', 'h5p',
['*', 'h6p', 'h6p', 'h6p']], ['*', 2.0, 'h1p', ['*', 'h3p', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
3.0, 'h1p', ['*', 'h3p', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'],
['*', 'j2p', 'j2p']], ['*', 'h1p', ['*', 'h3p', 'h3p', 'h3p'],
['*', 'h5p', 'h5p'], 'j2p'], ['*', 'h1p', ['*', 'h3p', 'h3p',
'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*',
17.0, 'h1p', ['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'h6p', ['*',
'j2p', 'j2p', 'j2p']], ['*', 23.0, 'h1p', ['*', 'h3p', 'h3p',
'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p']], ['*', 7.0, 'h1p',
['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'h6p', 'j2p'], ['*', 16.0,
'h1p', ['*', 'h3p', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 16.0, 'h1p', ['*', 'h3p', 'h3p',
'h3p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 2.0,
'h1p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p', 'h5p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 3.0, 'h1p', ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*', 'h1p',
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p', 'h5p'], 'j2p'], ['*',
5.0, 'h1p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 29.0, 'h1p', ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 37.0, 'h1p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'],
'h6p', ['*', 'j2p', 'j2p']], ['*', 13.0, 'h1p', ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 10.0, 'h1p',
['*', 'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p', 'j2p']], ['*', 46.0, 'h1p', ['*', 'h3p', 'h3p'],
'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
50.0, 'h1p', ['*', 'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 14.0, 'h1p', ['*', 'h3p', 'h3p'],
'h5p', ['*', 'h6p', 'h6p'], 'j2p'], ['*', 16.0, 'h1p', ['*',
'h3p', 'h3p'], ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 16.0, 'h1p', ['*', 'h3p', 'h3p'], ['*', 'h6p',
```

```
'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 4.0, 'h1p', 'h3p',
['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 10.0, 'h1p', 'h3p', ['*', 'h5p', 'h5p', 'h5p'],
'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*', 8.0, 'h1p', 'h3p',
['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*',
2.0, 'h1p', 'h3p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p', 'j2p'],
['*', 'h1p', 'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 14.0, 'h1p',
'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p', 'j2p']], ['*', 38.0, 'h1p', 'h3p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
38.0, 'h1p', 'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 13.0, 'h1p', 'h3p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p'], 'j2p'], ['*', 9.0, 'h1p', 'h3p',
'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 25.0, 'h1p', 'h3p', 'h5p', ['*', 'h6p', 'h6p',
'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 23.0, 'h1p', 'h3p',
'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*',
7.0, 'h1p', 'h3p', 'h5p', ['*', 'h6p', 'h6p', 'h6p'], 'j2p'],
['*', 'h1p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 4.0, 'h1p',
['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p', 'j2p']], ['*', 6.0, 'h1p', ['*', 'h5p', 'h5p',
'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
4.0, 'h1p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 'h1p', ['*', 'h5p', 'h5p', 'h5p'],
['*', 'h6p', 'h6p'], 'j2p'], ['*', 'h1p', ['*', 'h5p', 'h5p'],
['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 4.0, 'h1p', ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 6.0,
'h1p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 4.0, 'h1p', ['*', 'h5p', 'h5p'],
['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 'h1p',
['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*',
3.0, ['*', 'h2p', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h5p',
['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h2p', 'h2p', 'h2p'],
['*', 'h3p', 'h3p'], 'h5p', 'j2p'], ['*', ['*', 'h2p', 'h2p',
'h2p'], ['*', 'h3p', 'h3p'], 'h5p'], ['*', 4.0, ['*', 'h2p',
'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h6p', ['*', 'j2p', 'j2p']],
['*', 4.0, ['*', 'h2p', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h6p',
'j2p'], ['*', 3.0, ['*', 'h2p', 'h2p', 'h2p'], 'h3p', ['*', 'h5p',
'h5p'], ['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h2p', 'h2p',
'h2p'], 'h3p', ['*', 'h5p', 'h5p'], 'j2p'], ['*', ['*', 'h2p',
'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p']], ['*', ['*', 'h2p',
'h2p', 'h2p'], 'h3p', 'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 6.0, ['*', 'h2p', 'h2p', 'h2p'], 'h3p', 'h5p', 'h6p', ['*',
'j2p', 'j2p']], ['*', 7.0, ['*', 'h2p', 'h2p', 'h2p'], 'h3p',
'h5p', 'h6p', 'j2p'], ['*', 2.0, ['*', 'h2p', 'h2p', 'h2p'],
'h3p', 'h5p', 'h6p'], ['*', 4.0, ['*', 'h2p', 'h2p', 'h2p'],
'h3p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 4.0, ['*',
'h2p', 'h2p', 'h2p'], 'h3p', ['*', 'h6p', 'h6p'], 'j2p'], ['*',
['*', 'h2p', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p']], ['*', 3.0, ['*', 'h2p', 'h2p', 'h2p'],
['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 3.0, ['*',
'h2p', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*',
['*', 'h2p', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], 'h6p'], ['*',
['*', 'h2p', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 3.0, ['*', 'h2p', 'h2p', 'h2p'],
'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 3.0,
```

```
['*', 'h2p', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p'], 'j2p'],
['*', ['*', 'h2p', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p']],
['*', 3.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h5p',
['*', 'j2p', 'j2p', 'j2p']], ['*', 7.0, ['*', 'h2p', 'h2p'], ['*',
'h3p', 'h3p', 'h3p'], 'h5p', ['*', 'j2p', 'j2p']], ['*', 5.0,
['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'j2p'],
['*', ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h5p'],
['*', 4.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h6p',
['*', 'j2p', 'j2p', 'j2p']], ['*', 8.0, ['*', 'h2p', 'h2p'], ['*',
'h3p', 'h3p', 'h3p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 4.0,
['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p', 'h3p'], 'h6p', 'j2p'],
['*', 6.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'], ['*', 'h5p',
'h5p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 11.0, ['*', 'h2p',
'h2p'], ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], ['*', 'j2p',
'j2p']], ['*', 9.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p'], 'j2p'], ['*', 2.0, ['*', 'h2p', 'h2p'], ['*',
'h3p', 'h3p'], ['*', 'h5p', 'h5p']], ['*', ['*', 'h2p', 'h2p'],
['*', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 13.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*', 22.0, ['*',
'h2p', 'h2p'], ['*', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p',
'j2p']], ['*', 17.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
'h5p', 'h6p', 'j2p'], ['*', 3.0, ['*', 'h2p', 'h2p'], ['*',
'h3p', 'h3p'], 'h5p', 'h6p'], ['*', 8.0, ['*', 'h2p', 'h2p'],
['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 16.0, ['*', 'h2p', 'h2p'], ['*', 'h3p', 'h3p'],
['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 8.0, ['*', 'h2p',
'h2p'], ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'], 'j2p'], ['*',
3.0, ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p', 'h5p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 7.0, ['*', 'h2p', 'h2p'], 'h3p',
['*', 'h5p', 'h5p', 'h5p'], ['*', 'j2p', 'j2p']], ['*', 5.0,
['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p', 'h5p'], 'j2p'],
['*', ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p', 'h5p']],
['*', 2.0, ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p'], 'h6p',
['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 12.0, ['*', 'h2p',
'h2p'], 'h3p', ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p',
'j2p']], ['*', 22.0, ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p',
'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 16.0, ['*', 'h2p',
'h2p'], 'h3p', ['*', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 4.0,
['*', 'h2p', 'h2p'], 'h3p', ['*', 'h5p', 'h5p'], 'h6p'], ['*',
2.0, ['*', 'h2p', 'h2p'], 'h3p', 'h5p', ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 14.0, ['*', 'h2p',
'h2p'], 'h3p', 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 25.0, ['*', 'h2p', 'h2p'], 'h3p', 'h5p', ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 16.0, ['*', 'h2p',
'h2p'], 'h3p', 'h5p', ['*', 'h6p', 'h6p'], 'j2p'], ['*', 3.0,
['*', 'h2p', 'h2p'], 'h3p', 'h5p', ['*', 'h6p', 'h6p']], ['*',
4.0, ['*', 'h2p', 'h2p'], 'h3p', ['*', 'h6p', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 8.0, ['*', 'h2p', 'h2p'], 'h3p',
['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 4.0, ['*',
'h2p', 'h2p'], 'h3p', ['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*',
['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h2p', 'h2p'],
['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']],
['*', 6.0, ['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p', 'h5p'], 'h6p',
['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h2p', 'h2p'], ['*', 'h5p',
'h5p', 'h5p'], 'h6p', 'j2p'], ['*', ['*', 'h2p', 'h2p'], ['*',
'h5p', 'h5p', 'h5p'], 'h6p'], ['*', 2.0, ['*', 'h2p', 'h2p'],
['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
```

```
'j2p', 'j2p']], ['*', 8.0, ['*', 'h2p', 'h2p'], ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
12.0, ['*', 'h2p', 'h2p'], ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p'], ['*', 'j2p', 'j2p']], ['*', 8.0, ['*', 'h2p', 'h2p'],
['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], 'j2p'], ['*', 2.0, ['*',
'h2p', 'h2p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p']], ['*',
['*', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h2p', 'h2p'],
'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']],
['*', 6.0, ['*', 'h2p', 'h2p'], 'h5p', ['*', 'h6p', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 4.0, ['*', 'h2p', 'h2p'], 'h5p',
['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*', ['*', 'h2p', 'h2p'],
'h5p', ['*', 'h6p', 'h6p', 'h6p']], ['*', 6.0, 'h2p', ['*', 'h3p',
'h3p', 'h3p'], ['*', 'h5p', 'h5p'], ['*', 'j2p', 'j2p', 'j2p']],
['*', 11.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'], ['*', 'h5p',
'h5p'], ['*', 'j2p', 'j2p']], ['*', 6.0, 'h2p', ['*', 'h3p',
'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'j2p'], ['*', 'h2p', ['*',
'h3p', 'h3p', 'h3p'], ['*', 'h5p', 'h5p']], ['*', -1.0, ['*',
'h2p', ['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p',
'j2p', 'j2p', 'j2p']]], ['*', 17.0, 'h2p', ['*', 'h3p', 'h3p',
'h3p'], 'h5p', 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*', 25.0,
'h2p', ['*', 'h3p', 'h3p', 'h3p'], 'h5p', 'h6p', ['*', 'j2p',
'j2p']], ['*', 7.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'], 'h5p',
'h6p', 'j2p'], ['*', 16.0, 'h2p', ['*', 'h3p', 'h3p', 'h3p'],
['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 16.0,
'h2p', ['*', 'h3p', 'h3p', 'h3p'], ['*', 'h6p', 'h6p'], ['*',
'j2p', 'j2p']], ['*', 6.0, 'h2p', ['*', 'h3p', 'h3p'], ['*',
'h5p', 'h5p', 'h5p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 11.0,
'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p', 'h5p'], ['*',
'j2p', 'j2p']], ['*', 6.0, 'h2p', ['*', 'h3p', 'h3p'], ['*',
'h5p', 'h5p', 'h5p'], 'j2p'], ['*', 'h2p', ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p', 'h5p']], ['*', 3.0, 'h2p', ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 25.0, 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p',
'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*', 39.0, 'h2p',
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p',
'j2p']], ['*', 20.0, 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h5p',
'h5p'], 'h6p', 'j2p'], ['*', 3.0, 'h2p', ['*', 'h3p', 'h3p'],
['*', 'h5p', 'h5p'], 'h6p'], ['*', 6.0, 'h2p', ['*', 'h3p',
'h3p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 42.0, 'h2p', ['*', 'h3p', 'h3p'], 'h5p', ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 50.0, 'h2p',
['*', 'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 14.0, 'h2p', ['*', 'h3p', 'h3p'], 'h5p', ['*',
'h6p', 'h6p'], 'j2p'], ['*', 16.0, 'h2p', ['*', 'h3p', 'h3p'],
['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*',
16.0, 'h2p', ['*', 'h3p', 'h3p'], ['*', 'h6p', 'h6p', 'h6p'],
['*', 'j2p', 'j2p']], ['*', 4.0, 'h2p', 'h3p', ['*', 'h5p',
'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*',
14.0, 'h2p', 'h3p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p']], ['*', 18.0, 'h2p', 'h3p', ['*', 'h5p',
'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 10.0, 'h2p',
'h3p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 2.0,
'h2p', 'h3p', ['*', 'h5p', 'h5p', 'h5p'], 'h6p'], ['*', 'h2p',
'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 13.0, 'h2p', 'h3p', ['*',
'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 36.0, 'h2p', 'h3p', ['*', 'h5p', 'h5p'], ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 40.0, 'h2p',
```

```
'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 19.0, 'h2p', 'h3p', ['*', 'h5p', 'h5p'], ['*',
'h6p', 'h6p'], 'j2p'], ['*', 3.0, 'h2p', 'h3p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p']], ['*', 7.0, 'h2p', 'h3p', 'h5p',
['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']],
['*', 21.0, 'h2p', 'h3p', 'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 21.0, 'h2p', 'h3p', 'h5p', ['*',
'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 7.0, 'h2p',
'h3p', 'h5p', ['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*', 'h2p',
['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p', 'j2p', 'j2p']], ['*', 5.0, 'h2p', ['*', 'h5p',
'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 10.0, 'h2p', ['*', 'h5p', 'h5p', 'h5p'], ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 10.0, 'h2p',
['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 5.0, 'h2p', ['*', 'h5p', 'h5p', 'h5p'], ['*',
'h6p', 'h6p'], 'j2p'], ['*', 'h2p', ['*', 'h5p', 'h5p', 'h5p'],
['*', 'h6p', 'h6p']], ['*', 'h2p', ['*', 'h5p', 'h5p'], ['*',
'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p', 'j2p']],
['*', 5.0, 'h2p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 10.0, 'h2p', ['*',
'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p']], ['*', 10.0, 'h2p', ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 5.0, 'h2p', ['*',
'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'], 'j2p'], ['*', 'h2p',
['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p']], ['*', 6.0, ['*',
'h3p', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p',
'j2p', 'j2p', 'j2p']], ['*', 15.0, ['*', 'h3p', 'h3p', 'h3p'],
['*', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p']], ['*',
12.0, ['*', 'h3p', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], 'h6p',
['*', 'j2p', 'j2p']], ['*', 3.0, ['*', 'h3p', 'h3p', 'h3p'], ['*',
'h5p', 'h5p'], 'h6p', 'j2p'], ['*', 12.0, ['*', 'h3p', 'h3p',
'h3p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p',
'j2p']], ['*', 24.0, ['*', 'h3p', 'h3p', 'h3p'], 'h5p', ['*',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 12.0, ['*',
'h3p', 'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p']], ['*', 6.0, ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p',
'h5p'], 'h6p', ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 15.0,
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p', 'h5p'], 'h6p', ['*',
'j2p', 'j2p', 'j2p']], ['*', 12.0, ['*', 'h3p', 'h3p'], ['*',
'h5p', 'h5p', 'h5p'], 'h6p', ['*', 'j2p', 'j2p']], ['*', 3.0,
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p', 'h5p'], 'h6p', 'j2p'],
['*', 3.0, ['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], ['*', 'h6p',
'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 21.0,
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 39.0, ['*', 'h3p',
'h3p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p',
'j2p', 'j2p']], ['*', 27.0, ['*', 'h3p', 'h3p'], ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 6.0,
['*', 'h3p', 'h3p'], ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'],
'j2p'], ['*', 12.0, ['*', 'h3p', 'h3p'], 'h5p', ['*', 'h6p',
'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 24.0,
['*', 'h3p', 'h3p'], 'h5p', ['*', 'h6p', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p']], ['*', 12.0, ['*', 'h3p', 'h3p'], 'h5p',
['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*', 3.0,
'h3p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*',
'j2p', 'j2p', 'j2p', 'j2p', 'j2p']], ['*', 12.0, 'h3p', ['*',
'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p',
'j2p', 'j2p']], ['*', 18.0, 'h3p', ['*', 'h5p', 'h5p', 'h5p'],
```

```
['*', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p']], ['*', 12.0,
'h3p', ['*', 'h5p', 'h5p', 'h5p'], ['*', 'h6p', 'h6p'], ['*',
'j2p', 'j2p']], ['*', 3.0, 'h3p', ['*', 'h5p', 'h5p', 'h5p'],
['*', 'h6p', 'h6p'], 'j2p'], ['*', 3.0, 'h3p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p',
'j2p', 'j2p']], ['*', 12.0, 'h3p', ['*', 'h5p', 'h5p'], ['*',
'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p', 'j2p', 'j2p']], ['*',
18.0, 'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'],
['*', 'j2p', 'j2p', 'j2p']], ['*', 12.0, 'h3p', ['*', 'h5p',
'h5p'], ['*', 'h6p', 'h6p', 'h6p'], ['*', 'j2p', 'j2p']], ['*',
3.0, 'h3p', ['*', 'h5p', 'h5p'], ['*', 'h6p', 'h6p', 'h6p'],
'j2p']]],
['+', ['*', ['*', 'h1n', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'],
'h5n', ['*', 'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n', 'h1n'],
['*', 'h3n', 'h3n'], 'h5n', 'j2n'], ['*', 4.0, ['*', 'h1n', 'h1n',
'h1n'], ['*', 'h3n', 'h3n'], 'h6n', ['*', 'j2n', 'j2n']], ['*',
4.0, ['*', 'h1n', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h6n',
'j2n'], ['*', ['*', 'h1n', 'h1n', 'h1n'], 'h3n', ['*', 'h5n',
'h5n'], ['*', 'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n', 'h1n'],
'h3n', 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 5.0,
['*', 'h1n', 'h1n', 'h1n'], 'h3n', 'h5n', 'h6n', ['*', 'j2n',
'j2n']], ['*', 2.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n', 'h5n',
'h6n', 'j2n'], ['*', 4.0, ['*', 'h1n', 'h1n', 'h1n'], 'h3n',
['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h1n',
'h1n', 'h1n'], 'h3n', ['*', 'h6n', 'h6n'], 'j2n'], ['*', ['*',
'h1n', 'h1n', 'h1n'], ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n',
'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n', 'h1n'], ['*', 'h5n',
'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n',
'h1n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']],
['*', 2.0, ['*', 'h1n', 'h1n', 'h1n'], 'h5n', ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n', 'h1n'], 'h5n',
['*', 'h6n', 'h6n'], 'j2n'], ['*', 5.0, ['*', 'h1n', 'h1n'],
'h2n', ['*', 'h3n', 'h3n'], 'h5n', ['*', 'j2n', 'j2n']], ['*',
6.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h3n', 'h3n'], 'h5n',
'j2n'], ['*', ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h3n', 'h3n'],
'h5n'], ['*', 12.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h3n',
'h3n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 12.0, ['*', 'h1n',
'h1n'], 'h2n', ['*', 'h3n', 'h3n'], 'h6n', 'j2n'], ['*', 5.0,
['*', 'h1n', 'h1n'], 'h2n', 'h3n', ['*', 'h5n', 'h5n'], ['*',
'j2n', 'j2n']], ['*', 2.0, ['*', 'h1n', 'h1n'], 'h2n', 'h3n',
['*', 'h5n', 'h5n'], 'j2n'], ['*', 3.0, ['*', 'h1n', 'h1n'],
'h2n', 'h3n', 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*',
16.0, ['*', 'h1n', 'h1n'], 'h2n', 'h3n', 'h5n', 'h6n', ['*',
'j2n', 'j2n']], ['*', 11.0, ['*', 'h1n', 'h1n'], 'h2n', 'h3n',
'h5n', 'h6n', 'j2n'], ['*', 2.0, ['*', 'h1n', 'h1n'], 'h2n',
'h3n', 'h5n', 'h6n'], ['*', 12.0, ['*', 'h1n', 'h1n'], 'h2n',
'h3n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 12.0,
['*', 'h1n', 'h1n'], 'h2n', 'h3n', ['*', 'h6n', 'h6n'], 'j2n'],
['*', 3.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h5n', 'h5n'], 'h6n',
['*', 'j2n', 'j2n', 'j2n']], ['*', 5.0, ['*', 'h1n', 'h1n'],
'h2n', ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*',
2.0, ['*', 'h1n', 'h1n'], 'h2n', ['*', 'h5n', 'h5n'], 'h6n',
'j2n'], ['*', 3.0, ['*', 'h1n', 'h1n'], 'h2n', 'h5n', ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 7.0, ['*',
'h1n', 'h1n'], 'h2n', 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 5.0, ['*', 'h1n', 'h1n'], 'h2n', 'h5n', ['*',
'h6n', 'h6n'], 'j2n'], ['*', ['*', 'h1n', 'h1n'], 'h2n', 'h5n',
['*', 'h6n', 'h6n']], ['*', ['*', 'h1n', 'h1n'], ['*', 'h3n',
'h3n', 'h3n'], 'h5n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 2.0,
```

```
['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n', 'h3n'], 'h5n', ['*',
'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n',
'h3n'], 'h5n', 'j2n'], ['*', 4.0, ['*', 'h1n', 'h1n'], ['*',
'h3n', 'h3n', 'h3n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*',
8.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n', 'h3n'], 'h6n', ['*',
'j2n', 'j2n']], ['*', 4.0, ['*', 'h1n', 'h1n'], ['*', 'h3n',
'h3n', 'h3n'], 'h6n', 'j2n'], ['*', 2.0, ['*', 'h1n', 'h1n'],
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], ['*', 'j2n', 'j2n',
'j2n']], ['*', 4.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n'], ['*', 'j2n', 'j2n']], ['*', 2.0, ['*', 'h1n',
'h1n'], ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'j2n'], ['*',
['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h5n', 'h6n', ['*',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 12.0, ['*', 'h1n', 'h1n'],
['*', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']],
['*', 25.0, ['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], 'h5n',
'h6n', ['*', 'j2n', 'j2n']], ['*', 14.0, ['*', 'h1n', 'h1n'],
['*', 'h3n', 'h3n'], 'h5n', 'h6n', 'j2n'], ['*', 8.0, ['*', 'h1n',
'h1n'], ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n']], ['*', 16.0, ['*', 'h1n', 'h1n'], ['*', 'h3n',
'h3n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 8.0,
['*', 'h1n', 'h1n'], ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'],
'j2n'], ['*', ['*', 'h1n', 'h1n'], 'h3n', ['*', 'h5n', 'h5n',
'h5n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n'],
'h3n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'j2n', 'j2n']], ['*',
2.0, ['*', 'h1n', 'h1n'], 'h3n', ['*', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 11.0, ['*', 'h1n', 'h1n'],
'h3n', ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']],
['*', 13.0, ['*', 'h1n', 'h1n'], 'h3n', ['*', 'h5n', 'h5n'],
'h6n', ['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h1n', 'h1n'],
'h3n', ['*', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 2.0, ['*', 'h1n',
'h1n'], 'h3n', 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n', 'j2n']], ['*', 14.0, ['*', 'h1n', 'h1n'], 'h3n', 'h5n',
['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 26.0,
['*', 'h1n', 'h1n'], 'h3n', 'h5n', ['*', 'h6n', 'h6n'], ['*',
'j2n', 'j2n']], ['*', 14.0, ['*', 'h1n', 'h1n'], 'h3n', 'h5n',
['*', 'h6n', 'h6n'], 'j2n'], ['*', 4.0, ['*', 'h1n', 'h1n'],
'h3n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']],
['*', 8.0, ['*', 'h1n', 'h1n'], 'h3n', ['*', 'h6n', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h1n', 'h1n'], 'h3n', ['*',
'h6n', 'h6n', 'h6n'], 'j2n'], ['*', ['*', 'h1n', 'h1n'], ['*',
'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n', 'j2n']],
['*', 2.0, ['*', 'h1n', 'h1n'], ['*', 'h5n', 'h5n', 'h5n'], 'h6n',
['*', 'j2n', 'j2n', 'j2n']], ['*', ['*', 'h1n', 'h1n'], ['*',
'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 2.0,
['*', 'h1n', 'h1n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 6.0, ['*', 'h1n',
'h1n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n']], ['*', 6.0, ['*', 'h1n', 'h1n'], ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 2.0,
['*', 'h1n', 'h1n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
'j2n'], ['*', ['*', 'h1n', 'h1n'], 'h5n', ['*', 'h6n', 'h6n',
'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 3.0, ['*',
'h1n', 'h1n'], 'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n']], ['*', 3.0, ['*', 'h1n', 'h1n'], 'h5n', ['*',
'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', ['*', 'h1n',
'h1n'], 'h5n', ['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*', 7.0,
'h1n', ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h5n', ['*',
'j2n', 'j2n']], ['*', 9.0, 'h1n', ['*', 'h2n', 'h2n'], ['*',
'h3n', 'h3n'], 'h5n', 'j2n'], ['*', 2.0, 'h1n', ['*', 'h2n',
```

```
'h2n'], ['*', 'h3n', 'h3n'], 'h5n'], ['*', 12.0, 'h1n', ['*',
'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h6n', ['*', 'j2n', 'j2n']],
['*', 12.0, 'h1n', ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
'h6n', 'j2n'], ['*', 7.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n',
['*', 'h5n', 'h5n'], ['*', 'j2n', 'j2n']], ['*', 6.0, 'h1n',
['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n'], 'j2n'], ['*',
'h1n', ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n']], ['*',
3.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', 'h5n', 'h6n', ['*', 'j2n',
'j2n', 'j2n']], ['*', 17.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n',
'h5n', 'h6n', ['*', 'j2n', 'j2n']], ['*', 16.0, 'h1n', ['*',
'h2n', 'h2n'], 'h3n', 'h5n', 'h6n', 'j2n'], ['*', 4.0, 'h1n',
['*', 'h2n', 'h2n'], 'h3n', 'h5n', 'h6n'], ['*', 12.0, 'h1n',
['*', 'h2n', 'h2n'], 'h3n', ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 12.0, 'h1n', ['*', 'h2n', 'h2n'], 'h3n', ['*',
'h6n', 'h6n'], 'j2n'], ['*', 3.0, 'h1n', ['*', 'h2n', 'h2n'],
['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*',
7.0, 'h1n', ['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], 'h6n',
['*', 'j2n', 'j2n']], ['*', 5.0, 'h1n', ['*', 'h2n', 'h2n'],
['*', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 'h1n', ['*', 'h2n',
'h2n'], ['*', 'h5n', 'h5n'], 'h6n'], ['*', 3.0, 'h1n', ['*',
'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n']], ['*', 8.0, 'h1n', ['*', 'h2n', 'h2n'], 'h5n', ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 7.0, 'h1n', ['*',
'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n'], 'j2n'], ['*', 2.0,
'h1n', ['*', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n']], ['*',
4.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n', 'h3n'], 'h5n', ['*', 'j2n',
'j2n', 'j2n']], ['*', 9.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n',
'h3n'], 'h5n', ['*', 'j2n', 'j2n']], ['*', 6.0, 'h1n', 'h2n',
['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'j2n'], ['*', 'h1n', 'h2n',
['*', 'h3n', 'h3n', 'h3n'], 'h5n'], ['*', 8.0, 'h1n', 'h2n', ['*',
'h3n', 'h3n', 'h3n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*',
16.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n', 'h3n'], 'h6n', ['*',
'j2n', 'j2n']], ['*', 8.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n',
'h3n'], 'h6n', 'j2n'], ['*', 8.0, 'h1n', 'h2n', ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
17.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'],
['*', 'j2n', 'j2n']], ['*', 12.0, 'h1n', 'h2n', ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], 'j2n'], ['*', 2.0, 'h1n', 'h2n',
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n']], ['*', 2.0, 'h1n',
'h2n', ['*', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n',
'j2n', 'j2n']], ['*', 25.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n'],
'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 47.0, 'h1n',
'h2n', ['*', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n']],
['*', 31.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n'], 'h5n', 'h6n',
'j2n'], ['*', 3.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n'], 'h5n',
'h6n'], ['*', 16.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n'], ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 32.0, 'h1n',
'h2n', ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 16.0, 'h1n', 'h2n', ['*', 'h3n', 'h3n'], ['*',
'h6n', 'h6n'], 'j2n'], ['*', 4.0, 'h1n', 'h2n', 'h3n', ['*',
'h5n', 'h5n', 'h5n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 6.0,
'h1n', 'h2n', 'h3n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'j2n',
'j2n']], ['*', 2.0, 'h1n', 'h2n', 'h3n', ['*', 'h5n', 'h5n',
'h5n'], 'j2n'], ['*', 4.0, 'h1n', 'h2n', 'h3n', ['*', 'h5n',
'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 23.0,
'h1n', 'h2n', 'h3n', ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n',
'j2n', 'j2n']], ['*', 36.0, 'h1n', 'h2n', 'h3n', ['*', 'h5n',
'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 21.0, 'h1n', 'h2n',
'h3n', ['*', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 4.0, 'h1n',
```

```
'h2n', 'h3n', ['*', 'h5n', 'h5n'], 'h6n'], ['*', 4.0, 'h1n',
'h2n', 'h3n', 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n', 'j2n']], ['*', 28.0, 'h1n', 'h2n', 'h3n', 'h5n', ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 51.0, 'h1n',
'h2n', 'h3n', 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']],
['*', 30.0, 'h1n', 'h2n', 'h3n', 'h5n', ['*', 'h6n', 'h6n'],
'j2n'], ['*', 3.0, 'h1n', 'h2n', 'h3n', 'h5n', ['*', 'h6n',
'h6n']], ['*', 8.0, 'h1n', 'h2n', 'h3n', ['*', 'h6n', 'h6n',
'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 16.0, 'h1n', 'h2n',
'h3n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*',
8.0, 'h1n', 'h2n', 'h3n', ['*', 'h6n', 'h6n', 'h6n'], 'j2n'],
['*', 2.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 6.0, 'h1n', 'h2n', ['*',
'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*',
6.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n',
'j2n']], ['*', 2.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n', 'h5n'],
'h6n', 'j2n'], ['*', 4.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n'],
['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*',
14.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n']], ['*', 18.0, 'h1n', 'h2n', ['*',
'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*',
10.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
'j2n'], ['*', 2.0, 'h1n', 'h2n', ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n']], ['*', 2.0, 'h1n', 'h2n', 'h5n', ['*', 'h6n', 'h6n',
'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 7.0, 'h1n',
'h2n', 'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n']], ['*', 9.0, 'h1n', 'h2n', 'h5n', ['*', 'h6n', 'h6n',
'h6n'], ['*', 'j2n', 'j2n']], ['*', 5.0, 'h1n', 'h2n', 'h5n',
['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*', 'h1n', 'h2n', 'h5n',
['*', 'h6n', 'h6n', 'h6n']], ['*', 2.0, 'h1n', ['*', 'h3n', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
3.0, 'h1n', ['*', 'h3n', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'],
['*', 'j2n', 'j2n']], ['*', 'h1n', ['*', 'h3n', 'h3n', 'h3n'],
['*', 'h5n', 'h5n'], 'j2n'], ['*', 'h1n', ['*', 'h3n', 'h3n',
'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*',
17.0, 'h1n', ['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'h6n', ['*',
'j2n', 'j2n', 'j2n']], ['*', 23.0, 'h1n', ['*', 'h3n', 'h3n',
'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n']], ['*', 7.0, 'h1n',
['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'h6n', 'j2n'], ['*', 16.0,
'h1n', ['*', 'h3n', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 16.0, 'h1n', ['*', 'h3n', 'h3n',
'h3n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 2.0,
'h1n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n', 'h5n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 3.0, 'h1n', ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n', 'h5n'], ['*', 'j2n', 'j2n']], ['*', 'h1n',
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n', 'h5n'], 'j2n'], ['*',
5.0, 'h1n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 29.0, 'h1n', ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']],
['*', 37.0, 'h1n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'],
'h6n', ['*', 'j2n', 'j2n']], ['*', 13.0, 'h1n', ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 10.0, 'h1n',
['*', 'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n', 'j2n']], ['*', 46.0, 'h1n', ['*', 'h3n', 'h3n'],
'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
50.0, 'h1n', ['*', 'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', 14.0, 'h1n', ['*', 'h3n', 'h3n'],
'h5n', ['*', 'h6n', 'h6n'], 'j2n'], ['*', 16.0, 'h1n', ['*',
'h3n', 'h3n'], ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
```

```
'j2n']], ['*', 16.0, 'h1n', ['*', 'h3n', 'h3n'], ['*', 'h6n',
'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 4.0, 'h1n', 'h3n',
['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 10.0, 'h1n', 'h3n', ['*', 'h5n', 'h5n', 'h5n'],
'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 8.0, 'h1n', 'h3n',
['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*',
2.0, 'h1n', 'h3n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n', 'j2n'],
['*', 'h1n', 'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 14.0, 'h1n',
'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n', 'j2n']], ['*', 38.0, 'h1n', 'h3n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
38.0, 'h1n', 'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', 13.0, 'h1n', 'h3n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n'], 'j2n'], ['*', 9.0, 'h1n', 'h3n',
'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 25.0, 'h1n', 'h3n', 'h5n', ['*', 'h6n', 'h6n',
'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 23.0, 'h1n', 'h3n',
'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*',
7.0, 'h1n', 'h3n', 'h5n', ['*', 'h6n', 'h6n', 'h6n'], 'j2n'],
['*', 'h1n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 4.0, 'h1n',
['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n', 'j2n']], ['*', 6.0, 'h1n', ['*', 'h5n', 'h5n',
'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
4.0, 'h1n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', 'h1n', ['*', 'h5n', 'h5n', 'h5n'],
['*', 'h6n', 'h6n'], 'j2n'], ['*', 'h1n', ['*', 'h5n', 'h5n'],
['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 4.0, 'h1n', ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 6.0,
'h1n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 4.0, 'h1n', ['*', 'h5n', 'h5n'],
['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 'h1n',
['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*',
3.0, ['*', 'h2n', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h5n',
['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h2n', 'h2n', 'h2n'],
['*', 'h3n', 'h3n'], 'h5n', 'j2n'], ['*', ['*', 'h2n', 'h2n',
'h2n'], ['*', 'h3n', 'h3n'], 'h5n'], ['*', 4.0, ['*', 'h2n',
'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h6n', ['*', 'j2n', 'j2n']],
['*', 4.0, ['*', 'h2n', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h6n',
'j2n'], ['*', 3.0, ['*', 'h2n', 'h2n', 'h2n'], 'h3n', ['*', 'h5n',
'h5n'], ['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h2n', 'h2n',
'h2n'], 'h3n', ['*', 'h5n', 'h5n'], 'j2n'], ['*', ['*', 'h2n',
'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n']], ['*', ['*', 'h2n',
'h2n', 'h2n'], 'h3n', 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']],
['*', 6.0, ['*', 'h2n', 'h2n', 'h2n'], 'h3n', 'h5n', 'h6n', ['*',
'j2n', 'j2n']], ['*', 7.0, ['*', 'h2n', 'h2n', 'h2n'], 'h3n',
'h5n', 'h6n', 'j2n'], ['*', 2.0, ['*', 'h2n', 'h2n', 'h2n'],
'h3n', 'h5n', 'h6n'], ['*', 4.0, ['*', 'h2n', 'h2n', 'h2n'],
'h3n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 4.0, ['*',
'h2n', 'h2n', 'h2n'], 'h3n', ['*', 'h6n', 'h6n'], 'j2n'], ['*',
['*', 'h2n', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n']], ['*', 3.0, ['*', 'h2n', 'h2n', 'h2n'],
['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 3.0, ['*',
'h2n', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*',
['*', 'h2n', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], 'h6n'], ['*',
['*', 'h2n', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 3.0, ['*', 'h2n', 'h2n', 'h2n'],
```

```
'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 3.0,
['*', 'h2n', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n'], 'j2n'],
['*', ['*', 'h2n', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n']],
['*', 3.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h5n',
['*', 'j2n', 'j2n', 'j2n']], ['*', 7.0, ['*', 'h2n', 'h2n'], ['*',
'h3n', 'h3n', 'h3n'], 'h5n', ['*', 'j2n', 'j2n']], ['*', 5.0,
['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'j2n'],
['*', ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h5n'],
['*', 4.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h6n',
['*', 'j2n', 'j2n', 'j2n']], ['*', 8.0, ['*', 'h2n', 'h2n'], ['*',
'h3n', 'h3n', 'h3n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 4.0,
['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n', 'h3n'], 'h6n', 'j2n'],
['*', 6.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'], ['*', 'h5n',
'h5n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 11.0, ['*', 'h2n',
'h2n'], ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], ['*', 'j2n',
'j2n']], ['*', 9.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n'], 'j2n'], ['*', 2.0, ['*', 'h2n', 'h2n'], ['*',
'h3n', 'h3n'], ['*', 'h5n', 'h5n']], ['*', ['*', 'h2n', 'h2n'],
['*', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 13.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 22.0, ['*',
'h2n', 'h2n'], ['*', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n',
'j2n']], ['*', 17.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
'h5n', 'h6n', 'j2n'], ['*', 3.0, ['*', 'h2n', 'h2n'], ['*',
'h3n', 'h3n'], 'h5n', 'h6n'], ['*', 8.0, ['*', 'h2n', 'h2n'],
['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n']], ['*', 16.0, ['*', 'h2n', 'h2n'], ['*', 'h3n', 'h3n'],
['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 8.0, ['*', 'h2n',
'h2n'], ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'], 'j2n'], ['*',
3.0, ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n', 'h5n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 7.0, ['*', 'h2n', 'h2n'], 'h3n',
['*', 'h5n', 'h5n', 'h5n'], ['*', 'j2n', 'j2n']], ['*', 5.0,
['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n', 'h5n'], 'j2n'],
['*', ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n', 'h5n']],
['*', 2.0, ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n'], 'h6n',
['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 12.0, ['*', 'h2n',
'h2n'], 'h3n', ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n',
'j2n']], ['*', 22.0, ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n',
'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 16.0, ['*', 'h2n',
'h2n'], 'h3n', ['*', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 4.0,
['*', 'h2n', 'h2n'], 'h3n', ['*', 'h5n', 'h5n'], 'h6n'], ['*',
2.0, ['*', 'h2n', 'h2n'], 'h3n', 'h5n', ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 14.0, ['*', 'h2n',
'h2n'], 'h3n', 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n']], ['*', 25.0, ['*', 'h2n', 'h2n'], 'h3n', 'h5n', ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 16.0, ['*', 'h2n',
'h2n'], 'h3n', 'h5n', ['*', 'h6n', 'h6n'], 'j2n'], ['*', 3.0,
['*', 'h2n', 'h2n'], 'h3n', 'h5n', ['*', 'h6n', 'h6n']], ['*',
4.0, ['*', 'h2n', 'h2n'], 'h3n', ['*', 'h6n', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 8.0, ['*', 'h2n', 'h2n'], 'h3n',
['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 4.0, ['*',
'h2n', 'h2n'], 'h3n', ['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*',
['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h2n', 'h2n'],
['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']],
['*', 6.0, ['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n', 'h5n'], 'h6n',
['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h2n', 'h2n'], ['*', 'h5n',
'h5n', 'h5n'], 'h6n', 'j2n'], ['*', ['*', 'h2n', 'h2n'], ['*',
'h5n', 'h5n', 'h5n'], 'h6n'], ['*', 2.0, ['*', 'h2n', 'h2n'],
```

```
['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n', 'j2n']], ['*', 8.0, ['*', 'h2n', 'h2n'], ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
12.0, ['*', 'h2n', 'h2n'], ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n'], ['*', 'j2n', 'j2n']], ['*', 8.0, ['*', 'h2n', 'h2n'],
['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], 'j2n'], ['*', 2.0, ['*',
'h2n', 'h2n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n']], ['*',
['*', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h2n', 'h2n'],
'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']],
['*', 6.0, ['*', 'h2n', 'h2n'], 'h5n', ['*', 'h6n', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', 4.0, ['*', 'h2n', 'h2n'], 'h5n',
['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*', ['*', 'h2n', 'h2n'],
'h5n', ['*', 'h6n', 'h6n', 'h6n']], ['*', 6.0, 'h2n', ['*', 'h3n',
'h3n', 'h3n'], ['*', 'h5n', 'h5n'], ['*', 'j2n', 'j2n', 'j2n']],
['*', 11.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'], ['*', 'h5n',
'h5n'], ['*', 'j2n', 'j2n']], ['*', 6.0, 'h2n', ['*', 'h3n',
'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'j2n'], ['*', 'h2n', ['*',
'h3n', 'h3n', 'h3n'], ['*', 'h5n', 'h5n']], ['*', -1.0, ['*',
'h2n', ['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n',
'j2n', 'j2n', 'j2n']]], ['*', 17.0, 'h2n', ['*', 'h3n', 'h3n',
'h3n'], 'h5n', 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 25.0,
'h2n', ['*', 'h3n', 'h3n', 'h3n'], 'h5n', 'h6n', ['*', 'j2n',
'j2n']], ['*', 7.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'], 'h5n',
'h6n', 'j2n'], ['*', 16.0, 'h2n', ['*', 'h3n', 'h3n', 'h3n'],
['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 16.0,
'h2n', ['*', 'h3n', 'h3n', 'h3n'], ['*', 'h6n', 'h6n'], ['*',
'j2n', 'j2n']], ['*', 6.0, 'h2n', ['*', 'h3n', 'h3n'], ['*',
'h5n', 'h5n', 'h5n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 11.0,
'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n', 'h5n'], ['*',
'j2n', 'j2n']], ['*', 6.0, 'h2n', ['*', 'h3n', 'h3n'], ['*',
'h5n', 'h5n', 'h5n'], 'j2n'], ['*', 'h2n', ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n', 'h5n']], ['*', 3.0, 'h2n', ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 25.0, 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n',
'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*', 39.0, 'h2n',
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n',
'j2n']], ['*', 20.0, 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h5n',
'h5n'], 'h6n', 'j2n'], ['*', 3.0, 'h2n', ['*', 'h3n', 'h3n'],
['*', 'h5n', 'h5n'], 'h6n'], ['*', 6.0, 'h2n', ['*', 'h3n',
'h3n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 42.0, 'h2n', ['*', 'h3n', 'h3n'], 'h5n', ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 50.0, 'h2n',
['*', 'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 14.0, 'h2n', ['*', 'h3n', 'h3n'], 'h5n', ['*',
'h6n', 'h6n'], 'j2n'], ['*', 16.0, 'h2n', ['*', 'h3n', 'h3n'],
['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*',
16.0, 'h2n', ['*', 'h3n', 'h3n'], ['*', 'h6n', 'h6n', 'h6n'],
['*', 'j2n', 'j2n']], ['*', 4.0, 'h2n', 'h3n', ['*', 'h5n',
'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*',
14.0, 'h2n', 'h3n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n']], ['*', 18.0, 'h2n', 'h3n', ['*', 'h5n',
'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 10.0, 'h2n',
'h3n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 2.0,
'h2n', 'h3n', ['*', 'h5n', 'h5n', 'h5n'], 'h6n'], ['*', 'h2n',
'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 13.0, 'h2n', 'h3n', ['*',
'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 36.0, 'h2n', 'h3n', ['*', 'h5n', 'h5n'], ['*',
```

```
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 40.0, 'h2n',
'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 19.0, 'h2n', 'h3n', ['*', 'h5n', 'h5n'], ['*',
'h6n', 'h6n'], 'j2n'], ['*', 3.0, 'h2n', 'h3n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n']], ['*', 7.0, 'h2n', 'h3n', 'h5n',
['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']],
['*', 21.0, 'h2n', 'h3n', 'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 21.0, 'h2n', 'h3n', 'h5n', ['*',
'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 7.0, 'h2n',
'h3n', 'h5n', ['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*', 'h2n',
['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n', 'j2n', 'j2n']], ['*', 5.0, 'h2n', ['*', 'h5n',
'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 10.0, 'h2n', ['*', 'h5n', 'h5n', 'h5n'], ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 10.0, 'h2n',
['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 5.0, 'h2n', ['*', 'h5n', 'h5n', 'h5n'], ['*',
'h6n', 'h6n'], 'j2n'], ['*', 'h2n', ['*', 'h5n', 'h5n', 'h5n'],
['*', 'h6n', 'h6n']], ['*', 'h2n', ['*', 'h5n', 'h5n'], ['*',
'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n', 'j2n']],
['*', 5.0, 'h2n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 10.0, 'h2n', ['*',
'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
'j2n']], ['*', 10.0, 'h2n', ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 5.0, 'h2n', ['*',
'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'], 'j2n'], ['*', 'h2n',
['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n']], ['*', 6.0, ['*',
'h3n', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n',
'j2n', 'j2n', 'j2n']], ['*', 15.0, ['*', 'h3n', 'h3n', 'h3n'],
['*', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n']], ['*',
12.0, ['*', 'h3n', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], 'h6n',
['*', 'j2n', 'j2n']], ['*', 3.0, ['*', 'h3n', 'h3n', 'h3n'], ['*',
'h5n', 'h5n'], 'h6n', 'j2n'], ['*', 12.0, ['*', 'h3n', 'h3n',
'h3n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n',
'j2n']], ['*', 24.0, ['*', 'h3n', 'h3n', 'h3n'], 'h5n', ['*',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 12.0, ['*',
'h3n', 'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n']], ['*', 6.0, ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n',
'h5n'], 'h6n', ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 15.0,
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n', 'h5n'], 'h6n', ['*',
'j2n', 'j2n', 'j2n']], ['*', 12.0, ['*', 'h3n', 'h3n'], ['*',
'h5n', 'h5n', 'h5n'], 'h6n', ['*', 'j2n', 'j2n']], ['*', 3.0,
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n', 'h5n'], 'h6n', 'j2n'],
['*', 3.0, ['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], ['*', 'h6n',
'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 21.0,
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 39.0, ['*', 'h3n',
'h3n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n',
'j2n', 'j2n']], ['*', 27.0, ['*', 'h3n', 'h3n'], ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 6.0,
['*', 'h3n', 'h3n'], ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'],
'j2n'], ['*', 12.0, ['*', 'h3n', 'h3n'], 'h5n', ['*', 'h6n',
'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 24.0,
['*', 'h3n', 'h3n'], 'h5n', ['*', 'h6n', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n']], ['*', 12.0, ['*', 'h3n', 'h3n'], 'h5n',
['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*', 3.0,
'h3n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*',
'j2n', 'j2n', 'j2n', 'j2n', 'j2n']], ['*', 12.0, 'h3n', ['*',
'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n',
```

```
'j2n', 'j2n']], ['*', 18.0, 'h3n', ['*', 'h5n', 'h5n', 'h5n'],
['*', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n']], ['*', 12.0,
'h3n', ['*', 'h5n', 'h5n', 'h5n'], ['*', 'h6n', 'h6n'], ['*',
'j2n', 'j2n']], ['*', 3.0, 'h3n', ['*', 'h5n', 'h5n', 'h5n'],
['*', 'h6n', 'h6n'], 'j2n'], ['*', 3.0, 'h3n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n',
'j2n', 'j2n']], ['*', 12.0, 'h3n', ['*', 'h5n', 'h5n'], ['*',
'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n', 'j2n', 'j2n']], ['*',
18.0, 'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'],
['*', 'j2n', 'j2n', 'j2n']], ['*', 12.0, 'h3n', ['*', 'h5n',
'h5n'], ['*', 'h6n', 'h6n', 'h6n'], ['*', 'j2n', 'j2n']], ['*',
3.0, 'h3n', ['*', 'h5n', 'h5n'], ['*', 'h6n', 'h6n', 'h6n'],
'j2n']]
]

Perform the following steps: First, analyze the example and detect
the underlying patterns. Finally, implement the logic and provide
the python code. Your code must be able to deal with negative
variables, too. So do not use logarithms. Make sure to place all
tensors on the same device that the x tensor is on.

Write a function `compute_result(formulas, x, inputs)` that
receives the formulas, the variables x and the mapping to variable
names as an input, and returns a tensor with the corresponding
results.
```

LLM Response

```
To solve this problem, we'll parse the formulas to extract
individual terms, compute the necessary variable powers and
products efficiently, and sum the terms to obtain the final result
for each formula. Here's the code:

```python
import torch
from collections import defaultdict, Counter

def compute_result(formulas, x, inputs):
    device = x.device
    batch_size = x.size(0)
    num_vars = len(inputs)

    # Map variable names to indices in x
    varname_to_index = {var_name: idx for idx, var_name in
enumerate(inputs)}

    terms_coefficients = []
    terms_exponents = []
    terms_formula_idx = []

    # Function to parse terms recursively
    def parse_term(node):
        if isinstance(node, float):
            return node, Counter()
        elif isinstance(node, str):
            return 1.0, Counter({node: 1})
        elif isinstance(node, list):
            op = node[0]
            args = node[1:]
```
```

```
            if op == '*':
                coeff = 1.0
                exponents = Counter()
                for arg in args:
                    c, e = parse_term(arg)
                    coeff *= c
                    exponents.update(e)
                return coeff, exponents
            else:
                raise ValueError(f"Unsupported operation '{op}' in
term")
        else:
            raise ValueError(f"Unsupported term node: {node}")

    # Function to parse formulas and collect terms
    def parse_formula(node, formula_idx, formula_coeff):
        if isinstance(node, list):
            op = node[0]
            args = node[1:]
            if op == '+' or op == '-':
                op_coeff = 1.0 if op == '+' else -1.0
                for arg in args:
                    parse_formula(arg, formula_idx, formula_coeff
* op_coeff)
            elif op == '*':
                coeff, exponents = parse_term(node)
                terms_coefficients.append(formula_coeff * coeff)
                terms_exponents.append(exponents)
                terms_formula_idx.append(formula_idx)
            else:
                raise ValueError(f"Unsupported operation '{op}' in
formula")
        else:
            coeff, exponents = parse_term(node)
            terms_coefficients.append(formula_coeff * coeff)
            terms_exponents.append(exponents)
            terms_formula_idx.append(formula_idx)

    # Parse each formula
    for formula_idx, formula in enumerate(formulas):
        if formula[0] == '-':
            formula_coeff = -1.0
            parse_formula(formula[2], formula_idx, formula_coeff)
        elif formula[0] == '+':
            formula_coeff = 1.0
            parse_formula(formula[1:], formula_idx, formula_coeff)
        else:
            raise ValueError(f"Unsupported formula structure:
{formula}")

    num_terms = len(terms_coefficients)

    # Collect all variable names
    all_var_names = set()
    for exponents in terms_exponents:
        all_var_names.update(exponents.keys())

    # Prepare exponents tensor
```

```
    exponents_tensor = torch.zeros((num_terms, num_vars),
dtype=torch.float32, device=device)
    for term_idx, exponents in enumerate(terms_exponents):
        for var_name, exponent in exponents.items():
            var_idx = varname_to_index[var_name]
            exponents_tensor[term_idx, var_idx] = exponent

    # Prepare coefficients tensor
    coefficients_tensor = torch.tensor(terms_coefficients,
dtype=torch.float32, device=device)

    # Prepare term formula indices tensor
    terms_formula_idx_tensor = torch.tensor(terms_formula_idx,
dtype=torch.long, device=device)

    # Compute x_powers
    x_expanded = x.unsqueeze(1)  # Shape: [batch_size, 1,
num_vars]
    exponents_expanded = exponents_tensor.unsqueeze(0)  # Shape:
[1, num_terms, num_vars]
    x_powers = torch.pow(x_expanded, exponents_expanded)  # Shape:
[batch_size, num_terms, num_vars]

    # Compute term values
    term_products = torch.prod(x_powers, dim=2)  # Shape:
[batch_size, num_terms]
    term_values = coefficients_tensor.unsqueeze(0) * term_products
 # Shape: [batch_size, num_terms]

    # Compute outputs
    num_formulas = len(formulas)
    outputs = torch.zeros((batch_size, num_formulas),
dtype=torch.float32, device=device)
    outputs.scatter_add_(1, terms_formula_idx_tensor.unsqueeze(0).expand(batch_size,
num_terms), term_values)

    return outputs
```

This function first parses the formulas to extract terms, coefficients, and variable exponents. It then computes the necessary powers of variables and calculates the term values efficiently using tensor operations. Finally, it sums the term values for each formula to produce the output tensor.

Usage Example:
```python
# Example inputs
formulas = [
    ['-', 0.0, ['+', ['*', 2.0, ['*', 'x1', 'x1'], 'x2'], ['*',
-1.0, 'x1', ['*', 'x2', 'x2']]]],
    ['+', ['*', 3.0, 'x1', 'x2']]
]
x = torch.tensor([[1.0, 2.0], [3.0, 4.0]])  # Shape: [batch_size,
num_vars]
inputs = ['x1', 'x2']

# Compute the result
outputs = compute_result(formulas, x, inputs)
```

```
print(outputs)
```

Note:
- This code assumes that all variables in the formulas are present in the `inputs` list and that there are no unsupported operations in the formulas.
- It efficiently handles the computation by vectorizing over the batch and terms, avoiding any for-loops over tensors.
- All tensors are placed on the same device as the input tensor `x`.

Feedback to the LLM

```
ValueError: Unsupported operation '['*', -1.0, ['*', 2.0, ['*',
'h1n', 'h1n', 'h1n'], ['*', 'h2n', 'h2n'], 'h3n']]' in formula
```

LLM Response

You're right; my previous code didn't correctly handle the nested structures and the '-' operator with multiple operands in the formulas. I'll update the code to properly parse and handle nested formulas, including '+' and '-' operators with multiple operands.

Here's the corrected and optimized code:

```python
import torch

def compute_result(formulas, x, inputs):
    device = x.device
    batch_size = x.size(0)
    num_vars = len(inputs)

    # Map variable names to indices in x
    varname_to_index = {var_name: idx for idx, var_name in
enumerate(inputs)}

    terms_coefficients = []
    terms_exponents = []
    terms_formula_idx = []

    # Function to parse nodes recursively and collect terms
    def parse_node(node):
        if isinstance(node, float):
            return [(node, {})]
        elif isinstance(node, str):
            return [(1.0, {node: 1})]
        elif isinstance(node, list):
            op = node[0]
            operands = node[1:]
            if op == '+':
                terms = []
                for operand in operands:
                    terms.extend(parse_node(operand))
                return terms
            elif op == '-':
                if len(operands) == 1:
                    # Unary minus
```

```
                    terms = parse_node(operands[0])
                    return [(-coeff, exponents) for coeff,
exponents in terms]
                elif len(operands) == 2:
                    # Binary subtraction
                    terms1 = parse_node(operands[0])
                    terms2 = parse_node(operands[1])
                    negated_terms2 = [(-coeff, exponents) for
coeff, exponents in terms2]
                    return terms1 + negated_terms2
                else:
                    raise ValueError(f"Unsupported '-' operation
with operands: {operands}")
            elif op == '*':
                # Compute product of terms
                terms = [(1.0, {})]
                for operand in operands:
                    operand_terms = parse_node(operand)
                    new_terms = []
                    for coeff1, exponents1 in terms:
                        for coeff2, exponents2 in operand_terms:
                            coeff = coeff1 * coeff2
                            exponents = exponents1.copy()
                            for var, exp in exponents2.items():
                                exponents[var] =
exponents.get(var, 0) + exp
                            new_terms.append((coeff, exponents))
                    terms = new_terms
                return terms
            else:
                raise ValueError(f"Unsupported operation '{op}'")
        else:
            raise ValueError(f"Unsupported node type: {node}")

    # Parse each formula
    for formula_idx, formula in enumerate(formulas):
        terms = parse_node(formula)
        for coeff, exponents in terms:
            terms_coefficients.append(coeff)
            terms_exponents.append(exponents)
            terms_formula_idx.append(formula_idx)

    num_terms = len(terms_coefficients)

    # Prepare exponents tensor
    exponents_tensor = torch.zeros((num_terms, num_vars),
dtype=torch.float32, device=device)
    for term_idx, exponents in enumerate(terms_exponents):
        for var_name, exponent in exponents.items():
            var_idx = varname_to_index[var_name]
            exponents_tensor[term_idx, var_idx] = exponent

    # Prepare coefficients tensor
    coefficients_tensor = torch.tensor(terms_coefficients,
dtype=torch.float32, device=device)

    # Prepare term formula indices tensor
    terms_formula_idx_tensor = torch.tensor(terms_formula_idx,
dtype=torch.long, device=device)
```

```
    # Compute x to the power of exponents
    x_expanded = x.unsqueeze(1)  # Shape: [batch_size, 1,
num_vars]
    exponents_expanded = exponents_tensor.unsqueeze(0)  # Shape:
[1, num_terms, num_vars]
    x_powers = torch.pow(x_expanded, exponents_expanded)  # Shape:
[batch_size, num_terms, num_vars]

    # Compute term values
    term_products = torch.prod(x_powers, dim=2)  # Shape:
[batch_size, num_terms]
    term_values = coefficients_tensor.unsqueeze(0) * term_products
 # Shape: [batch_size, num_terms]

    # Compute outputs
    num_formulas = len(formulas)
    outputs = torch.zeros((batch_size, num_formulas),
dtype=torch.float32, device=device)
    outputs.scatter_add_(1, terms_formula_idx_tensor.unsqueeze(0).expand(batch_size,
num_terms), term_values)

    return outputs
```

Explanation:

- The `parse_node` function recursively parses the formula tree and handles the nested operations correctly, including unary and binary '-' operators.
- It collects all terms with their coefficients and exponents.
- After collecting all terms, it constructs tensors for coefficients and exponents and computes the powers of variables efficiently using tensor operations.
- It avoids any for-loops over tensors to ensure high efficiency and leverages PyTorch's broadcast and vectorized operations.
- All computations are performed on the same device as the input tensor `x`.

Usage Example:

Let's test this function with an example similar to your provided formulas:

```python
import torch

# Example formulas
formulas = [
    ['-', 0.0, ['+', ['*', -1.0, ['*', 2.0, ['*', 'h1p', 'h1p',
'h1p'], ['*', 'h2p', 'h2p'], 'h3p']]]],
    ['+', ['*', 3.0, 'h1n', 'h2n']]
]

# Input tensor x (random values for illustration)
inputs = ['h1p', 'h2p', 'h3p', 'h1n', 'h2n']
batch_size = 4
x = torch.randn(batch_size, len(inputs))
```

```
# Compute the result
outputs = compute_result(formulas, x, inputs)
print(outputs)
```

This code will compute the results for the provided formulas without any errors, handling negative variables and nested operations properly.

Note:

- The function assumes that variable names in the formulas match exactly with those in the `inputs` list.
- The code efficiently computes the powers and products using tensor operations, ensuring high performance, especially on a GPU.
- Since the formulas can be complex and generate many terms, the function is designed to handle them efficiently without recomputing shared sub-terms.