**Lifted Inference**

Many AI problems in fields such as machine learning, network communication, computer vision, and robotics can be solved using graphical probability models. Although these inference solutions are quite elegant, they do not take advantage of the symmetries or redundancies which exist implicitly in the graph structure. This is referred to as “Grounded Inference”. If we do, however, acknowledge and utilize these symmetries or redundancies, we can speed up the solving process and figuratively lift it to a higher level (-> lifted).

An example of this would be if was a conditional dependency between on the one hand, Microsoft computers and their average age, and on the other hand Apple computers and their average age. Let’s assume both dependencies are the same. Traditional models would still need separate potential functions to declare these dependencies, while lifted models understand the symmetry and can generalize the problem to company and age. This makes processing the probability much more efficient.

Initially, lifted inference was proposed by David Poole to make reasoning algorithms high-level, but the focus quickly shifted towards machine learning purposes. Numerous papers were published around this topic, showing that the technique is actually viable and speeds up traditional learning and reasoning tasks. Now that it has become more mainstream, we can return to Poole’s original idea.

We will now discuss one of the latest additions to the lifted inference resume. It is a field of study which is often used as a benchmark to test the cognitive and logical problem-solving skills in humans; combinatorics.

**Combinatorics**

Combinatorics is the mathematical field that concerns with counting all possible arrangements of a group of objects, given certain constraints. For example: “In how many possible ways can a deck of cards be shuffled?”. To which the answer is 52! (52 factorial). One could add a constraint, for example “… if the first card has to be an ace of spades.”, after which the answer changes to 51!. These counting problems are closely related to probability, but have discrete solutions, rather than percentages. They can also be expressed as a counting constraint satisfaction problem (#CSP) as we will see later on.

Humans are quite good at solving combinatorics problems as they recognize commonly reoccurring structures in these problems for which there exist enclosed formulas.

**Twelvefold Way**

Before continuing, let’s discuss a general framework for characterizing the 12 most common counting problems between 2 finite sets, known as “The Twelvefold Way”. Each problem can be calculated using a mathematical formula. Between the mentioned sets exists a function f, which can be injective, surjective or neither. To understand this intuitively, let’s think of the domain set as a bag of balls and the image set as a collection of boxes. Then, a function is equivalent with putting every ball in one of the boxes. If every box must contain at least 1 ball, the function is surjective. If each box may only have at most 1 ball , the function is injective. When none of these criteria is met, the function is neither.

Next, we can also categorize the problems of The Twelvefold Way by distinguishability. The sets X and Y can be both distinguishable (**≠)**, only X can be distinguishable (and X indistinguishable ? ) (**=X**), only Y can be distinguishable(**=Y**), or they can be both indistinguishable (**=**). This grouping can again be explained using the balls & boxes example. If a set is said to be distinguishable, then the balls (X) or the boxes (Y) have properties that make them different from the rest. An easy way is to visualize this is to imagine the balls or boxes to have different colours. This way, putting a red ball in a box is a different case than putting a blue ball in set box. If a set is indistinguishable, however, then each ball (X) or box (Y) has the same monotone colour and putting ball A in box B is the same case as putting ball A in box C.

**Counting Constraint Satisfaction Problems**

A normal constraint satisfaction problem (CSP) is a tuple where V is a set of variables, D a domain and C a set of constraints. An assignment satisfies a CSP if it satisfies all the constraints in C. The goal is to determine if a satisfying assignment exists. A satisfying assignment is also called a solution or **model** for the problem . In a *counting* constraint satisfaction problem (#CSP), the task is to find the number of satisfying assignments . Any combinatorics problem can be expressed as a #CSP by encoding the set X as a set of variables and letting set Y be the domain.

-Lifted reasoning from probabilistic inference for model counting problems

**CoLa**

In order to model combinatorics problems, we can use the declarative language CoLa (Combinatorics Language). CoLa defines a number of object and constraint types which, when combined, can be used to describe any counting problem. Elements are atomic objects which can be counted. Domains are sets of elements. They group elements together, according to some common property. A domain of ‘student’ elements could be ‘French students’ for example. The domain which contains all possible elements in the scope of a given problem is called the universe set. Finally, structures represent the different problems from the twelvefold way. They consist of pairs (D, F) where D is the distinguishability and F is the function type.

Constraints include: domain formulas, choice constraints and counting constraints. Domain formulas describe any set operation performed on a domain or another domain formula. Choice constraints fix the position one or more elements, given a combinatorial structure. For example: the first student in a sequence must be French. Lastly, counting constraints limit the number of elements from a domain that can be included for a given case by boolean comparison operators (>, >=, <, <=, =).

The CoLa language can only be interpreted using the CoSo solver. This solver supports lifted reasoning and is faster in solving time than any other combinatorics solver, both grounded and lifted. (-> because of the new lifted reasoning methods, the solver is a proof of the optimality of the new methods, which other solvers do not use.)

**Lifted Reasoning Over Counting Problems**

Let’s talk about some lifted reasoning strategies for #CSP’s.

Exchangeability is an important concept for lifted reasoning, since you can reason over groups of variables and get exponential improvements as a result. In a CSP, a tuple of variables () are defined exchangeable if for all satisfying assignments () and all permutations of (1, … , n), {} is a satisfying assignment as well. Less formally, let’s imagine that we’re playing a coin flipping game where the players choose heads or tails and whichever gets tossed the most out of 5 times wins the game. The tuple of variables () represent the tosses and if you’re the player who chose heads, the constraint would be #heads > #tails. The model () would be an example of a satisfying assignment, but so is the model (). We can clearly see that the tosses are exchangeable, which in this case means that the order of the tosses do not matter.

-Constraint Shattering

**The Solver**

The efficacy of the new lifted inference concepts for #CSP’s can be verified with the implementation of CoSo, a solver for combinatorial problems that’s based on the aforementioned exchangeability and constraint shattering.

**Visualization**

We will be taking the aforementioned combinatorics solver to a new level. We will implement a graphical user interface which can be used to help students understand the solving mechanisms behind problems. The interface will be able to calculate the solution for a given combinatorics problem, using the CoLa language and the (lifted) CoSo solver. But more importantly, it will be able to show the different steps that are used in the solving process. The solver namely tackles more complicated counting problems by breaking them down into smaller problems, for which enclosed formulas exist. This is the same technique which humans use and therefore it will give all necessary steps for students to follow and to solve the problems on their own. (Symbolab example).

For the visualization we plan on using the Godot game engine. Godot uses a programming language called GDScript, which is closely related Python, the language of CoSo. With Godot, we will be able to execute OS commands to access external files like the CoSo solver or other python libraries.

For the representation of sets, we will be using area-proportional Venn diagrams. As the name suggests, these diagrams are constructed in a way such that the areas of the sets and their intersections are in proportion with their respective element sizes. As the math for this is rather complicated, we will be using a related python library.

**Glossary**

**Reasoning Algorithm:** generates conclusions from known facts by logical techniques (induction, deduction, …).

**Lifted Reasoning = Lifted Inference:** exploiting symmetries (redundancies) to speed up reasoning algorithms.

**Graphical Probability Model:** a model for a problem consisting of a graph, where the nodes are random variables and the edges represent conditional dependencies. Often used in machine learning.

https://en.wikipedia.org/wiki/Combinatorics

https://en.wikipedia.org/wiki/Twelvefold\_way