GWV – Grundlagen der Wissensverarbeitung
Tutorial 11: Practical Assignment

The aim of the following assignments is to deepen your understanding of knowledge processing with some practical experience. **Choose *one* of the following assignments**. The results will be presented in class. You can choose any programming language you like, but be aware that you will probably have to explain your program in more detail to fellow students during presentation if you use a more esoteric language.

Remarks:

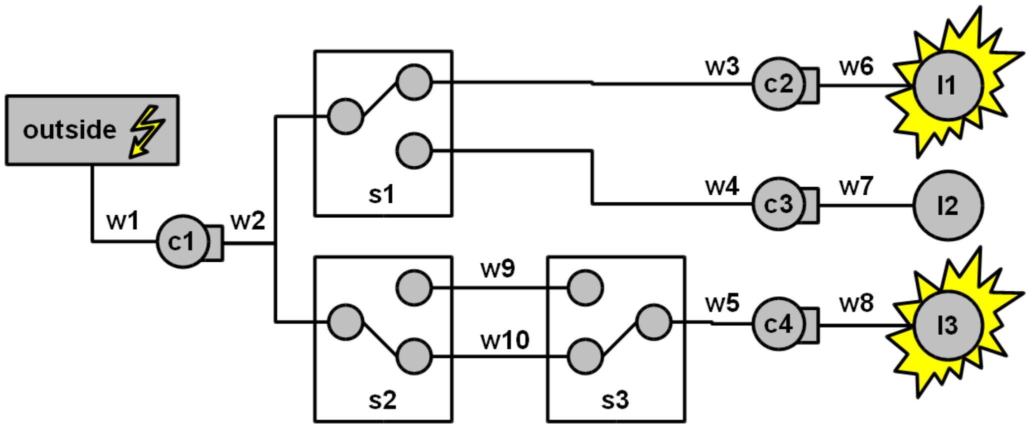
- Comment your code to guide other students through your implementation steps.
- Prepare a demonstration.
- Prepare an additional presentation up to 10 slides explaining the project topic.
- Every group hands out their own code and optional additional writings. **Strictly no working among the groups is allowed**.
- Every member of the group should be able to explain and present parts of the work.
- For most project suggestions it holds that a solution can be ***very straight forward but ineffective*** or ***very complex and intelligent***. Coming up with a very intelligent system is desirable but can lead to a big effort in terms of time spend for this project. Keep in mind to aim for a appropriate complexity and go for "optional" extensions only within reasonable time limits.

Exercise 11.1 : (Meta Interpreter and User Interaction)

of 36

This assignment consists of two parts: In the first part a knowledge base for the electrical domain (known from the lectures) is modeled using *prolog*. In the second part a *meta interpreter* that can take into account user queries is developed.

1. Part 1: Modeling the knowledge base:



The figure above shows the electrical wiring of a fictional building, the following abbreviations are used: l = lamp, w = wire, c = circuit breaker (Sicherung), s = switch. For the first part of this assignment, you can assume that all elements work as intended.

(a) Describe the depicted electrical wiring using the predicates: light(X), up(S), down(S), ok(E) and connectedTo(X,Y).

light(L) is true if L is a lamp.

up(S) is true if the switch S is in position up.

down(S) is true if the switch S is in position down.

ok(E) is true if E is not broken, E can be a lamp or a circuit breaker.

connectedTo(X,Y) is true if X is connected to Y so that electrical current can go from X to Y. In some cases the truth value of this predicate is depended on the position of a switch or the condition of a circuit breaker.

(b) An element of the electric wiring is supplied with power if connected to another element that is already supplied with power. Develop a transitive predicate live(W) to model this rule. For this assignment you can assume that live(outside) is true.

live(W) is true if W is supplied with power.

(c) A lamp is lit if supplied with power and not broken. Write a predicate lit(L).

lit(L) is true if L is lit.

(d) Finally, you can test your model and see if it can explain why lamp 1 and 3 are lit while lamp 2 remains dark.

2. Part 2: Meta interpreter

   (a) Write an a vanilla meta interpreter in PROLOG. *(Hint: You can find further information on this topic in chapter 6 of* Computational Intelligence *by Poole, Mackworth and Goebel.)*

   (b) Rewrite the knowledge base for the electrical domain for usage with your meta interpreter.

   (c) Test your meta interpreter with several queries.

   (d) Assuming that the electrical wiring does not change over time. Such an information could easily be coded in the knowledge base of an expert system. The position of switches and the condition of circuit-beakers and lamps is on the other hand subject to change. An artificial system can not know these facts for sure, but it could rely on the observations of a human observer. Rewrite both your meta interpreter and your knowledge base in a way that the position of switches and the condition of lamps and circuit breakers become atoms and that your meta interpreter asks the user for the truth value of these atoms when they are part of a proof.

   (e) Test your ask-the-user interpreter with several queries.

   (f) This assignment demonstrated how a human user can interact with a reasoning system by answering simple questions. Can you think of other forms of useful interaction?

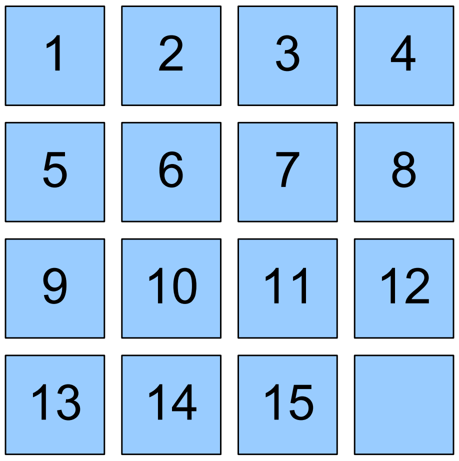
The challenge: Read up on the topic of meta interpreters on your own - this has not been covered fully in the lecture or or the assignments. You should only choose this assignment if you feel comfortable with programming in prolog.

Exercise 11.2 : (15-puzzle)

of 36

The 15-puzzle problem, consists of a 4 x 4 board with 15 sliding tiles, numbered 1 to 15, and one blank tile. There are four operators, which move the blank up, down, left, and right.

Write a program that can take any current state of the puzzle as an input and find the shortest way to solve the puzzle (if a solution exists). The puzzle is considered solved, when the tiles are arranged in the following way:



Use a heuristic search algorithm for the problem and devise a suitable heuristic function.

Your program should be able to do the following:

1. Accept input
2. Find a solution (if there is a solution)
3. Output the solution, so that it is human understandable (text output is fine)

The challenge: Though this assignment is straight forward, you will not be able to use uninformed search algorithms due to the size of the search space. When designing a heuristic for an informed search algorithm keep in mind that evaluating a heuristic function cost time too. Devise a heuristic and chose a search algorithm that can actually solve the given problem in a timely manner.

Optional: Turn your program into a game

1. Provide some form of GUI so that the user can slide the tiles around.
2. Generate random start configurations for the user to solve, but make sure these configurations can actually be solved.
3. Provide the user with a hint on his next step, if the user clicks a 'hint' button.
4. Judge the user's performance by comparing the number of steps the user took to solve the puzzle with the minimum number of steps necessary for solving the puzzle.

5. You can also offer more complex puzzles with more tiles (5 x 5 and more).
6. You can use a picture instead of numbers.

Exercise 11.3 : (Cryptoarithmetic Puzzle)

of 36

Cryptoarithmetic puzzles are a mental exercise that follow one simple rule: In a cryptoarithmetic puzzle each letter stands for a unique digit. The same letter stands for the same digit throughout the puzzle. Different digits are different letters. There are no leading zeros.

Here is an example:

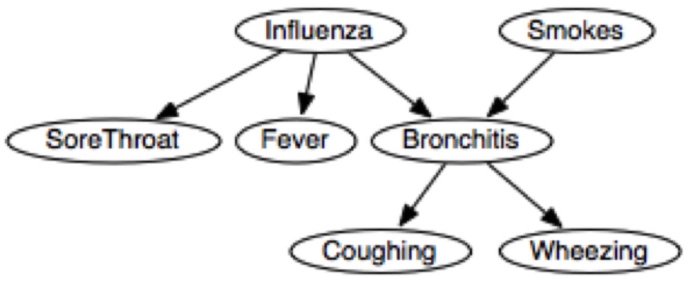
$$\begin{array}{r} \texttt{THREE} \\ + \ \texttt{THREE} \\ + \ \texttt{FIVE} \\ \hline \texttt{ELEVEN} \end{array} \quad \Longrightarrow \quad \begin{array}{r} 56711 \\ + \ 56711 \\ + \ 8491 \\ \hline 121913 \end{array}$$

These puzzles can be solved in a methodical way, using constraints and the AC-3 algorithm. Write a program, that can solve any cryptoarithmetic problem using constraints. Your program should be able to do the following:

1. Accept up to 5 input words.
   (It is always assumed your words line up to the right when doing addition.)
2. Find all possible solutions to the puzzle (if there is a solution).
3. Output the solution in a way that it is human understandable (text output is fine).
4. Test your algorithm against at least 5 different Cryptoarithmetic puzzles and report the results. *Hint: You can find a large database of these puzzles at:* `http://bach.istc.kobe-u.ac.jp/llp/crypt.html`
5. Use a word list to automatically generate new puzzles. Try to make your algorithm smarter than brute-force.

Exercise 11.4: (Diagnosis and Belief-Networks)

Suppose you want the diagnostic assistant to be able to reason about the possible causes of a patient's wheezing and coughing, as in figure 2. The agent can observe coughing, wheezing, and fever and can ask whether the patient smokes or has a sore throat. There are thus variables for these. There are variables that are useful to predict the outcomes of patients. The medical community has named many of these and characterized their symptoms. Here we will use the variables Bronchitis and Influenza. Now consider what the variables directly depend on. Whether patients wheeze depends on whether they have bronchitis. Whether they cough depends on whether they have bronchitis. Whether patients have bronchitis depends on whether they have influenza and whether they smoke. Whether they have fever depends on whether they have influenza. Figure 2 depicts these dependencies. Each of the variables will be Boolean, with domain true,false, representing the presence or absence of the associated disease or symptom. You assess how each variable depends on its parents, which is done by specifying the conditional probabilities of each variable given its parents:



$P(influenza) = 0.05$
$P(smokes) = 0.2$
$P(soreThroat \mid influenza) = 0.3$
$P(soreThroat \mid \neg influenza) = 0.001$
$P(fever \mid influenza) = 0.9$
$P(fever \mid \neg influenza) = 0.05$
$P(bronchitis \mid influenza \land smokes) = 0.99$
$P(bronchitis \mid influenza \land \neg smokes) = 0.9$
$P(bronchitis \mid \neg influenza \land smokes) = 0.7$
$P(bronchitis \mid \neg influenza \land \neg smokes) = 0.0001$
$P(coughing \mid bronchitis) = 0.8$
$P(coughing \mid \neg bronchitis) = 0.07$
$P(wheezing \mid bronchitis) = 0.6$
$P(wheezing \mid \neg bronchitis) = 0.001$

The process of diagnosis is carried out by conditioning on the observed symptoms and deriving posterior probabilities of the faults or diseases.

Your program should do the following:

1. Accept input representing observations on the variables: Sore throat, fever, coughing, wheezing, smokes.

2. Compute posterior probability of unobserved variable influenza.
3. Test your program against the following 3 cases:
   - Patient 1 has a sore throat (Nothing else is known about the patient.)
   - Patient 2 has a sore throat and is a smoker. (Nothing else is known about the patient.)
   - Patient 3 has a sore throat an coughs. (Nothing else is known about the patient.)
4. How can a diagnosis agent make suggestions about which observation would be most helpful in refining the diagnosis. (Consider for example patient 3. Is it more helpful to take his temperature or to auscultate his lung for example?)
   - Not all observable variables are needed to compute the posterior probability of influenza. But the more observable variables are used, the more accurate the diagnosis will be. Given any patient a medic can do the following actions to make further observations.
     - Ask if the patient smokes. (smokes)
     - Take the patients temperature (fever)
     - Check if the patient coughs (coughing)
     - Auscultate the patient's lung (wheezing)

Exercise 11.5 : (Yahtzee (Kniffel): Utility and Game AI)

of
36

Yahtzee (Kniffel) is a popular game that is mainly about decision under uncertainty. If you are not familiar with the game you can find the rules at: `www.hasbro.com/common/instruct/Yahtzee.pdf`.

You goal is to develop an agent that is able to achieve high scores in Yahtzee games. (Once you become more familiar with the strategies of the game, you will find that winning against a single opponent and just trying to get a very high score might call for slightly different strategies.) You should use utility networks to make informed decision under uncertainty that the dice roles will bring to the game.

Hint: Depending on your design, you might find that computing all expected utilities for each decision at each step of the game might be too complex. You can (and probably should) use simplification and heuristics to make the problem easier to handle. You could for instance apply greedy strategies that just optimize the outcome of a single turn, not the overall game. But at least at some point you should actually calculate utilities.

Exercise 11.6: (Author identification)

Sometimes we are faced with texts that lack author information. We can nonetheless try to guess the author based on other texts.

For this assignment you have a set of comments without author information as well as a set with author information. Write a program that guesses the author based on the annotated comments.

You do this by generating a language model for each author and then for each author computing the probability that he/she wrote the comments in question.

You can find the data on the GWV wiki.

Exercise 11.7: (Willikins-Seating-Problem (Optimization & Local Search))

of 36

Once again, Willikins (the butler of Sir Samuel of Ankh) is tasked with the rather complex challenge of finding the best seating order around the table. This time, instead of doing everything by hand again, he is searching for an easier solution.

Write a programm, that solves the Willikins-Seating-Problem:

- Input: list of persons, relationship rating between persons
- Output: optimized seating order

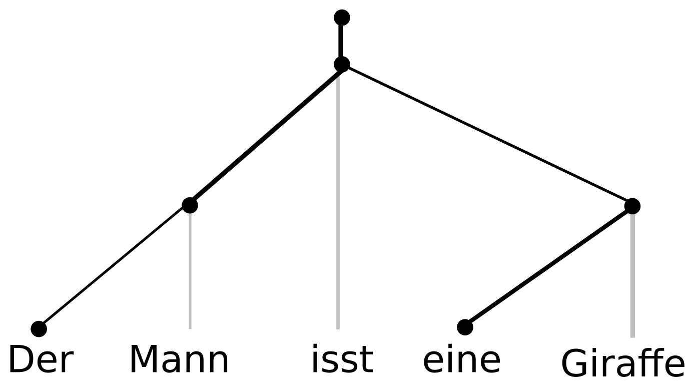
Remark: Unlike the class exercise 8.1. there is no fixed number of people sitting at the table. Your program should solve the Willkins-Seating-Problem for any number of people. Furthermore, the stopping criterion for the optimization algorithms is the time or steps spent, because there is no knowledge about the optimal evaluation score.

1. Design and implement the representation for the problem. There should be an additional option to randomize and save the relationship rating between persons, so you can generate bigger/different problems faster by only inputting a list of names. You can generate the names randomly on various websites.
2. Implement a brute force algorithm (random assignment every step), so you can compare its performance to the optimization algorithms.
3. Implement a greedy ascent algorithm with at least two different selection methods (discussed in Chapter 4.8.1). *Hint: Deciding fast can sometimes be more beneficial than evaluating many possible options. Try out both extremes.*
4. Extend the greedy ascent algorithm (as a separate function) with random restart and random walk. Additionally, save the best seating order to avoid losing it by random moves. The amount of randomness should be easily adjustable.
5. Choose and implement one of the other remaining optimization algorithms discussed in the book that you find interesting.
6. Evaluate the algorithms with run-time distributions (see chapter 4.8.3). Don't forget to use different problem sizes (e.g. 10, 50, 100 people) and different options (randomness, selection method, time). How do these algorithms compare? Write down your findings thoroughly. You can also plot the results with matplotlib for an easier overview.

Exercise 11.8 : (Dependency Parsing (Search and HMMs))

of
36

In tutorial 9, you implemented a Part-of-Speech (PoS) tagger using Hidden Markov Models and in tutorial 6, you got introduced to dependency parsing. The parser you read about was able to create a dependency tree for the sentence “Der Mann isst die Giraffe”. To be able to generalize for the similar sentence “Der Hund isst die Giraffe”, the parser’s rules must not rely on the words itself, like “Mann”, but on the words’ PoS tag.



1. Reuse the code of your PoS tagger as a base for this project.
2. Implement a program that is able to extract rules for the dependency parser from given training data. The training data contains sentences and their corresponding dependency trees.
3. Implement the parser you read about in tutorial 6.
4. Evaluate the performance of your parser on given test data.

Hint: Your group supervisor will supply you with suitable training and test data for the project.

Version: January 9, 2017
Achievable score on this sheet: 288