ARTIFICIAL INTELLIGENCE REPORT

PREPARED BY

Aman Kumar(20GG20004)
Shriyam Pandey(20GG20042)
Gaurav Shresth(20BT10018)

AI approach to Clinical Practice Guidelines

AI in medical science

PRESENT

Artificial Intelligence (AI) has already made a huge impact on our current technological trends in the medical health domain. The idea of using an AI system to diagnose and provide remedies for a person's daily minor health issues could save tons of time and money of visiting and waiting for doctors at clinics.

Artificial intelligence in medicine is the use of machine learning models to search medical data and uncover insights to help improve health outcomes and patient experiences.

Currently, the most common roles for AI in medical settings are clinical decision support and imaging analysis. Clinical decision support tools help providers make decisions about treatments, medications, mental health and other patient needs by providing them with quick access to information or research that's relevant to their patient. In medical imaging, AI tools are being used to analyze CT scans, x-rays, MRIs and other images for lesions or other findings that a human radiologist might miss.

Some ways in which AI positively impacts the practice of medicine, and helps clinicians make better decisions are listed as follows.

Informed patient care
Integrating medical AI into clinician workflows can give providers valuable context while they're making care decisions. A trained machine learning algorithm can help cut down on research time by giving clinicians valuable search results with evidence-based insights about treatments and procedures while the patient is still in the room with them.

Error reduction
A recent systemic review of 53 peer-reviewed studies examining the impact of AI on patient safety found that AI-powered decision support tools can help improve error detection and drug management.

Reducing the costs
There are a lot of potential ways AI could reduce costs across the healthcare industry.

Increased doctor-patient engagement
Many patients think of questions outside of typical business hours. AI can help provide around-the-clock support through chatbots that can answer basic questions and give patients resources when their provider's office isn't open. AI could also potentially be used to triage questions and flag information for further review, which could help alert providers to health changes that need additional attention.

CLINICAL PRACTICE GUIDELINES

Clinical practice guidelines are systematically developed statements to assist practitioners and patient decisions about appropriate health care for specific clinical circumstances which can be implemented through the AI.

Attribute	Value
Dyspnea	exertional
Beta Agonist	good response
Difficult Speech	absent
Tachycardia	absent
Typical Episode	worse

	Mild
Dyspnea	exertional
Beta Agonist	good response
Difficult Speech	absent
Tachycardia	absent
Tachypnea	exertional
Accessory muscles	none
Breathing sounds	normal or reduced
Typical episode	better or same
Sa02	>95
PEV	>75
FEV	>75

An important application of AI in medical science is to facilitate a doctor with important data and diagnoses which can help in making care decisions.

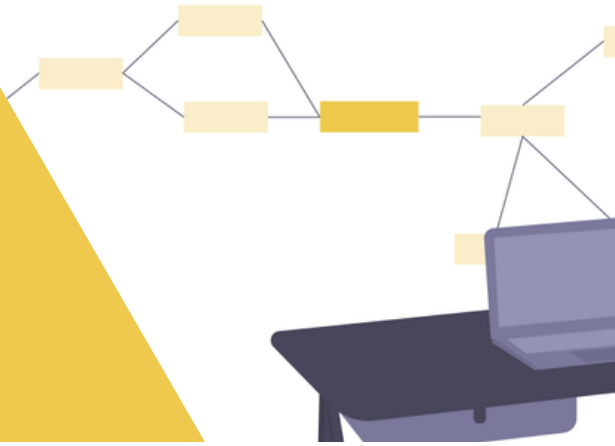
Modeling a CPG through AI can overcome limitations of current guideline models such as missing or mismatched data.

.

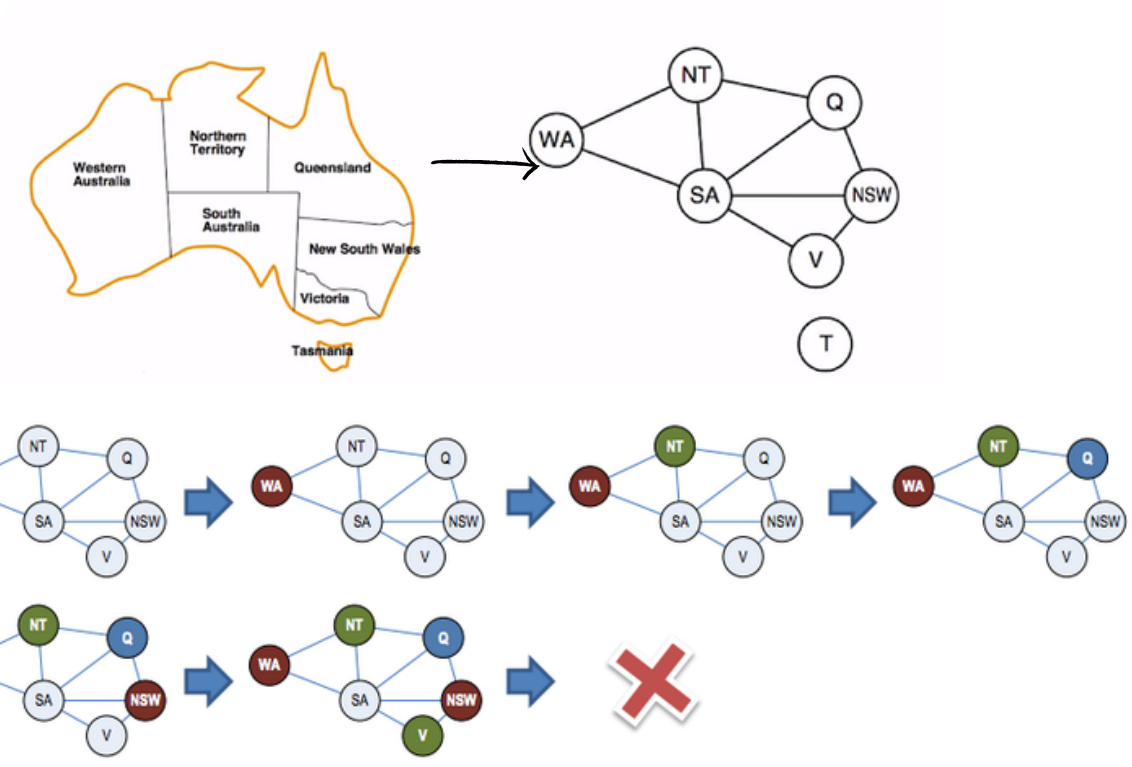
In this project, we implant a flexible data-driven approach to CPG implementation, where a CPG is cast as a constraint satisfaction model with constraints defining allowable variable values and permitted combinations of values for clinical variables from a CPG.

Incomplete clinical data is input to a constraint model and extended to complete CPG solutions for describing a patient state.

INTRODUCTION TO CSP



Constraint satisfaction solves problems by stating constraints about the problem area and consequently finding solutions that satisfy all or some of these constraints. A constraint satisfaction problem is defined by a tuple P = (X, D, C) where X = {X1,.....,Xn} is a finite set of variables, each associated with a domain of discrete values D = {D1,.....,Dn}, and a set of constraints C = {C1,....Cn}. Each constraint Ci is expressed by a relation Ri on some subset of variable values, Ri ⊆Di1 × ...x Dik and denotes the tuples that satisfy Ci5. A solution to a constraint satisfaction problem is an assignment of domain values to variables, in such a way that constraints are satisfied and solutions are found by systematically searching through the space of possible assignments of values to variables. This may involve finding just one solution with no preferences, all solutions or an optimal solution given some objective function determined in terms of some or all variables.



PROBLEM STATEMENT

The given project is to determine the severity of any disease (DDS which stands of Diagnosis Decision Step) through the input data of various health attributes given for particular medical conditions

Mathematical Modelling:

Using the constraint satisfaction approach outcome of a DDS from a CPG is modeled as a tuple CPG_D DSi = (Xi, Di, Ci), where Xi = {Xi1,…, Xin} is a finite set of clinical attributes, Di={Di1,…., Din}, are the domains of the attributes, i.e. a set of attribute values and Ci = {Ci1,…, Cin} is a set of constraints for DDS (which may be mild, moderate, or severe).

We'll try to find the most suitable DDS for the given input data which may be inconsistent/incomplete/complete values of different health attributes (CPG). In case of conflict between different DDS and input attributes, we'll try to rank different DDS on the basis of the Euclidean distance between input attributes and conflicting DDS.

$$d(x, dds_j) = \sqrt{\sum_i (x_i - dds_{ji})^2}$$

$$x - \text{input attributes}$$
$$dds_j - \text{attributes for particular DDS}$$

AI MODELING

Constraints:
We begin by describing constraints for different cases of input data.

The first types of constraints are referred to as partial-patient-descriptor-constraints and define all possible values for individual clinical attributes in a DDS. A unary constraint for a DDSi is:

$$Ci = \{Xi\{Zi\}\}, \text{ where } Zi \subseteq Di \text{ and } Zi \neq \varphi$$

In order to ensure node consistency, these unary constraints are used to sanity check individual attribute values.

Second, there is the complete patient-descriptor constraint. Constraints also model relationships between all possible attribute values for a given DDS (combined with a Boolean AND) and are used to determine viable complete solutions for describing a patient state. The constraints of a DDSi are:

$$Ci=\{Xi1\{Zi\}\wedge Xi2\{Zi\}\wedge....\wedge Xin(Zi)\}, \text{ where } Zi\subseteq Di \text{ and } Zi\neq\varphi$$

AI MODELING

Step 1: Perform node consistency check on incomplete input solution - A confirmed diagnosis and an incomplete solution (set of partial clinical data) provided by a physician as he/she assesses a patient is an input to the constraint satisfaction model. The DDS constraint model matching the diagnosis supplied by the physician is invoked and a node-consistency check is performed on each value from the incomplete input using the set of partial patient descriptor constraints. There are three possible outcomes to the node consistency check – no consistent nodes are found, all nodes are found to be consistent, or the result may be a combination of consistent and inconsistent nodes.

- If no consistent nodes (health attributes) are found (i.e. all partial-patient-descriptor constraints are violated), the physician is informed that all data entered is inconsistent for the confirmed diagnosis.

- If all nodes are consistent (i.e. all partial-patient-descriptor-constraints are satisfied), the incomplete solution may be extended to a feasible solution for the particular DDS. Therefore the input solution is retained as is and provided as an input to the complete patient descriptor constraints.

- Finally, if a combination of consistent and inconsistent nodes are found (i.e. some partial patient-descriptor constraints are violated and some are satisfied), inconsistent values are eliminated from the incomplete solution and consistent values are retained. Inconsistent values are referred to as conflict values and are flagged to draw attention to the mismatched CPG and patient data in final solutions. Retained values that have satisfied partial-patient-descriptor-constraints are provided as input to the complete patient-descriptor-constraints.

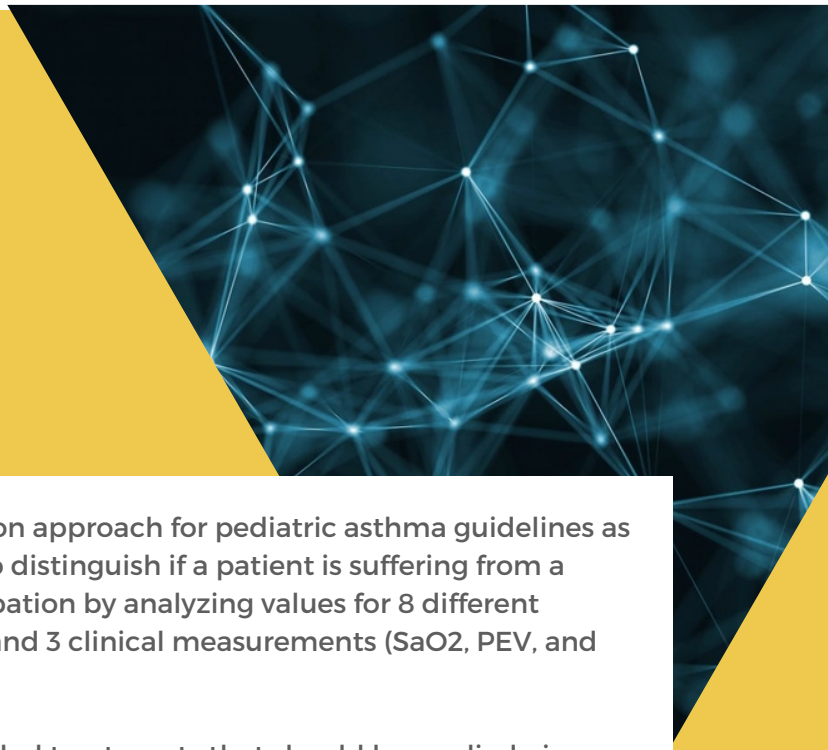
AI MODELING

Step 2: Extend incomplete solution to complete solution - Given a node consistent input solution obtained using step 1, the approach extends the incomplete solution to complete solutions for describing the patient state. Complete solutions are those which contain feasible values for all possible clinical attributes for a DDS from the CPG. Solutions are found using a backtracking search and complete patient-descriptor constraints. Given the incomplete input, values for missing variables are found that are consistent with values present in the current input solution; variables are instantiated sequentially and when all variables are instantiated, the validity of the constraint is checked using complete-patient descriptor-constraints. The search continues until all possible solutions are found.

Step 3: Order complete solutions - A DDS constraint model may return multiple outcomes or feasible solutions for an input solution. Therefore, solutions are ranked using a normalized Euclidean distance measure to calculate the similarity between an infeasible input solution (containing conflict values) and feasible solutions returned by the DSS model (where conflict values were replaced by missing values). The measure requires non-numeric attribute values in the DDS model (where conflict values were replaced by missing values) to be encoded in a numeric format. In our execution, we are performing steps 1 and 3 at simultaneously.

Sample Implementation: Pediatric Asthma

Execution



We demonstrate our constraint satisfaction approach for pediatric asthma guidelines as shown. The guideline allows physicians to distinguish if a patient is suffering from a mild, moderate, or severe asthma exacerbation by analyzing values for 8 different clinical signs (dyspnea - typical episode) and 3 clinical measurements (SaO2, PEV, and FEV).

The guideline also consists of recommended treatments that should be applied given different levels of exacerbation severity, however, our CPG constraint modeling approach currently focuses only on the decision step component of guideline modeling, and thus, we omit the guideline component that deals with treatment. To transform the asthma CPG from its representation in Table to one consistent with our constraint satisfaction approach, we define a DDS constraint model for each diagnosis/severity category; mild, moderate, and severe.

For each model the set of variables X, are the 11 clinical signs and measurements, X = {dyspnea, beta-agonist,..., FEV} as shown in the first column of Table, and D, the set of domain values are the associated values for those signs and measurements in the columns labeled mild, moderate and severe in Table.

	Mild	Moderate	Severe
Dyspnea	exertional	at rest	labored
Beta Agonist	good response	partial response	weak or no response
Difficult Speech	absent	absent or moderate	moderate or present
Tachycardia	absent	absent	present
Tachypnea	exertional	exertional or at rest	at rest or labored
Accessory muscles	none	none or moderate	moderate or severe
Breathing sounds	normal or reduced	reduced	reduced or silent
Typical episode	better or same	same or worse	same or worse
Sa02	>95	92–95	<92
PEV	>75	50–75	<50
FEV	>75	50–75	<50

Execution

Variables

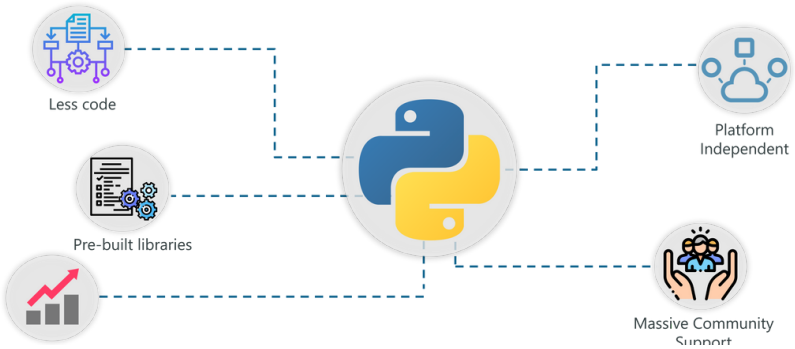
X= { 'Dyspnea', 'Beta Agonist', 'Difficult Speech', 'Tachycardia', 'Tachypnea', 'Accessory muscles', 'Breathing sounds', 'Typical episode', 'Sa02', 'PEV', 'FEV' }

Domain

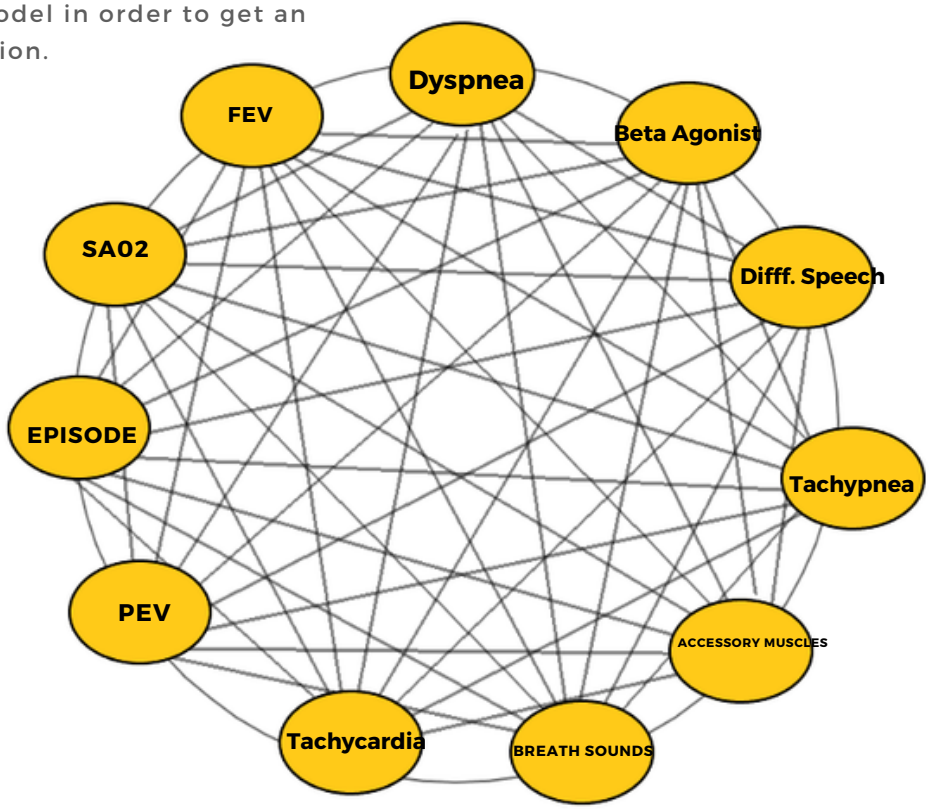
Dyspnea (X1) - { exertional, at rest, labored}
Beta Agonist (X2) - { good response, partial response, weak, no response}
Difficult Speech (X3) - {absent, moderate, present}
Tachycardia (X4) - { absent, present}
Tachypnea (X5) - { exertional, at rest, labored }
Accessory muscles (X6) - { none, moderate, severe }
Breathing sounds (X7) - { normal, reduced, silent }
Typical episode (X8) - { better, same, worse }
Sa02 (X9) - { 80-100 }
PEV (X10) - { 40-100}
FEV (X11) - { 40-100 }

FOR THE MODERATE DDS THE VARIABLE X3 = {DIFFICULT SPEECH} HAS DOMAIN VALUES D3 = {ABSENT, MODERATE}.

SOLUTION APPROACH



For the execution of our project, we are using the python programming language. The problem is framed as a Constraint Satisfaction model in order to get an optimum solution.



Constraint graph for pediatric asthma

The execution of the project through python has the following significant parts

1. The domain of each Diagnosis(Variable) in a CPG
2. Partial-Patient-Descriptor-Constraints(ppdc)
3. Complete-Patient-Descriptor-Constraint(cpcd)
4. Encoding of cpdc attribute values to numeric values
5. Disease CPG input
6. Node consistency check on incomplete input solution
7. The incomplete solution to complete solution

PYTHON STEP

Variables: All the keys of the domain dictionary are the variable for the Asthma CPG model.

{'dyspnea', 'beta agonist', 'difficult speech', 'tachycardia', 'tachypnea', 'accessory muscles', 'breathing sounds', 'typical episode', 'sao2', 'pev', 'fev'}

Domain: For defining the domain of all attributes, we used a dictionary datatype with attributes as keys and list of respective domain value.

In ndomain we assign values to each non numeric attributes to use them in Partial-Patient-Descriptor-Constraints

Domian of each Diagnosis(Variable) in a CPG

```
[ ]  domain = {"dyspnea": ["exertional","at rest","labored"],
               "beta agonist": ["good response","partial response","weak response","no response"],
               "difficult speech": ["absent","moderate","present"],
               "tachycardia": ["absent","present"],
               "tachypnea": ["exertional","at rest","labored"],
               "accessory muscles": ["severe","moderate","none"],
               "breathing sounds": ["normal","reduced","silent"],
               "typical episode": ["better","same","worse"],
               "sao2": list(range(80,100)),
               "pev": list(range(40,100)),
               "fev": list(range(40,100))
              }
     ndomain = {"dyspnea": [11,22,33],
               "beta agonist": [11,22,33,33],
               "difficult speech": [12,23,33],
               "tachycardia": [12,33],
               "tachypnea": [12,23,33],
               "accessory muscles": [33,23,12],
               "breathing sounds": [11,13,33],
               "typical episode": [12,13,33],
               "sao2": list(range(80,100)),
               "pev": list(range(40,100)),
               "fev": list(range(40,100))
              }
```

PYTHON STEP

As discussed above, For each DDS model we define two sets of constraints:

1. Partial-patient-descriptor-constraints
2. Complete-patient-descriptor-constraints

Partial-patient-descriptor-constraints: For defining all the unary constraints, we used a dictionary of dictionary. The first key is attribute value and second dictonary keys denote their severity/stage.

Partial-Patient-Descriptor-Constraints (ppdc)

```
ppdc = {"dyspnea":{"DDS1":["exertional"],
                   "DDS2":["at rest"],
                   "DDS3":["labored"]
                  },
        "beta agonist":{"DDS1":["good response"],
                        "DDS2":["partial response"],
                        "DDS3":["weak","no response"]
                       },
        "difficult speech":{"DDS1":["absent"],
                            "DDS2":["absent","moderate"],
                            "DDS3":["moderate","present"]
                           },
        "tachycardia":{"DDS1":["absent"],
                       "DDS2":["absent"],
                       "DDS3":["present"]
                      },
        "tachypnea":{"DDS1":["exertional"],
                     "DDS2":["exertional","at rest"],
                     "DDS3":["at rest","labored"]
                    },
        "accessory muscles":{"DDS1":["none"],
                             "DDS2":["none","moderate"],
                             "DDS3":["moderate","severe"]
                            },
        "breathing sounds":{"DDS1":["normal","reduced"],
                            "DDS2":["reduced"],
                            "DDS3":["reduced","silent"]
                           },
        "typical episode":{"DDS1":["better","same"],
                           "DDS2":["same","worse"],
                           "DDS3":["same","worse"]
                          },
        "sao2":{"DDS1":list(range(96,100)),
                "DDS2":list(range(76,100)),
                "DDS3":list(range(76,100))
               },
        "pev":{"DDS1":list(range(92,95)),
               "DDS2":list(range(50,75)),
               "DDS3":list(range(50,75))
              },
        "fev":{"DDS1":list(range(0,91)),
               "DDS2":list(range(0,50)),
               "DDS3":list(range(0,50))
              }
        }
```

PYTHON STEP

In the following code, we declare the complete-patient-descriptor-constraint and their numeric values, using nested dictonaries.

Complete-Patient-Descriptor-Constraint (cpdc)

```
cpdc = {"DDS1" :{"dyspnea":["exertional"],"beta agonist":["good response"],
                 "difficult speech":["absent"],"tachycardia":["absent"],"tachypnea":["exertional"]
                 ,"accessory muscles":["none"],"breathing sounds": ["normal","reduced"]
                 ,"typical episode": ["better","same"],"sao2":list(range(96,100)),
                 "pev": list(range(76,100)),"fev": list(range(76,100))},
        "DDS2" :{"dyspnea":["at rest"],"beta agonist":["partial response"],
                 "difficult speech":["absent","moderate"]
                 ,"tachycardia":["absent"],"tachypnea":["exertional","at rest"],
                 "accessory muscles":["none","moderate"],"breathing sounds": ["reduced"],
                 "typical episode": ["same","worse"],"sao2":list(range(92,95)),
                 "pev": list(range(50,75)),"fev": list(range(50,75))},
        "DDS3" :{"dyspnea":["labored"],"beta agonist":["weak","no response"],
                 "difficult speech":["moderate","present"],"tachycardia":["present"]
                 ,"tachypnea":["at rest","labored"],"accessory muscles":["moderate","severe"],
                 "breathing sounds": ["reduced","silent"]
                 ,"typical episode": ["same","worse"],"sao2":list(range(80,91)),
                 "pev": list(range(40,50)),"fev": list(range(40,50))},
       }
```

Encoded cpdc attribute values to numeric values

```
ncpdc = {"DDS1" :{"dyspnea":[11],"beta agonist":[11],
                  "difficult speech":[12],"tachycardia":[12],
                  "tachypnea":[12],"accessory muscles":[12],
                  "breathing sounds": [11,13],"typical episode": [11,13],
                  "sao2":list(range(96,100)),"pev": list(range(92,95)),
                  "fev": list(range(80,91))},
         "DDS2" :{"dyspnea":[22],"beta agonist":[22],"difficult speech":[12,23],
                  "tachycardia":[12],"tachypnea":[12,23],"accessory muscles":[12,23],
                  "breathing sounds": [13],"typical episode": [13,23],
                  "sao2":list(range(76,100)),"pev": list(range(50,75)),
                  "fev": list(range(40,50))},
         "DDS3" :{"dyspnea":[33],"beta agonist":[33,33],
                  "difficult speech":[23,33],"tachycardia":[33],
                  "tachypnea":[23,33],"accessory muscles":[23,33],
                  "breathing sounds": [13,33],"typical episode": [13,23],
                  "sao2":list(range(76,100)),"pev": list(range(50,75)),
                  "fev": list(range(40,50))},
        }
```

PYTHON STEP

Pediatric Asthma CPG input

```
class CPGInput:

  def __init__(self):
    self.att = {}

  def add(self):
    print("Choose the best suited option and if you don't know then choose \'NO\'")

    attributes = domain.keys()
    for attribute in attributes:
      print(attribute," : ",domain[attribute])
      self.key = attribute
      self.value = input("Enter :")
      self.att[self.key] = self.value
      print("\n")

    print("Based of the above input, what do you think physician in which state of asthma patient is \n[ mild , moderate , severe ] :")
    state = input()
    if state == "mild":
      self.state = "DDS1"
    elif state == "moderate":
      self.state = "DDS2"
    elif state == "severe":
      self.state = "DDS3"
```

We have defined a class here to take input of the incomplete/ inconsistent state of the patient suffering from Pediatric Asthma and a diagonsis from the doctor. Next we check consistency of a node using ppdc and conflicting values are flagged with 1, consistant with 2 and no data, with 0.

Node consistency check on incomplete input solution -

```
def check1(pState):
  flag = []
  cnt = 0
  attributes = domain.keys()
  state = pState.state

  for attribute in attributes:
    if pState.att[attribute] == "NO" or pState.att[attribute] == "no":
      flag.append(0)
      continue

    else:
      cnt = cnt+1
      for value in ppdc[attribute][state]:
        if pState.att[attribute] == value :
          present = True

      if present:
        flag.append(2)

      else:
        flag.append(1)

  if cnt == flag.count(1):
    print("All data entered is inconsistent for the confirmed diagnosis.")
  else:
    check2(pState,flag)
```

PYTHON STEP

Incomplete solution to complete solution -

```
def check2(pState,flag):
  state = pState.state

  if flag.count(1) == 0:
    attributes = list(domain.keys())
    for (conflict,attribute) in zip(flag,attributes):
      if conflict == 0:
        index = ncpdc[state][attribute].index(min(ncpdc[state][attribute]))
        pState.att[attribute] = cpdc[state][attribute][index]

  else:
    attributes = list(domain.keys())
    for (conflict,attribute) in zip(flag,attributes):
      if conflict == 1:
        value = minEuclideanDist(pState,attribute,pState.att[attribute])
        pState.att[attribute] = value

      elif conflict == 0:
        index = ncpdc[state][attribute].index(max(ncpdc[state][attribute]))
        pState.att[attribute] = cpdc[state][attribute][index]


  solution(pState,flag)
```

The check2( ) function has been defined to check for conflicting values and assign values to incomplete attributes. It checks the flag list to see if there are conflicting values.

If there is no conflicting value in the given input, then it will select all the unfilled attributes one by one and assign the most appropriate and consistent value for the confirmed state.

If there is/are conflicting values, the code will iterate over all conflicting and incomplete values, and for conflicting values, it will explore the respective domain and assign the value to the attribute where the Euclidean distance is the closest.

PYTHON STEP

minEuclideanDist: Returns the closest value or minimum Euclidean distance
def euclideanDist: used to find euclidean distance,
solution: to collect and print the solution ,
printDiagnosis: Print all attributes along with their values,
printState: printing the confirmed state
the check2 function utilizes all of them.

```
[ ]  def minEuclideanDist(pState,attribute,conflict):
       state = pState.state
       nconflict = ndomain[attribute][domain[attribute].index(conflict)]
       similarity = []

       constraintList = ncpdc[attribute][state]
       for constraint in constraintList:
         distance = euclideanDist(constraint,nconflict)
         similarity.append(distance)

       value = min(similarity)
       index = ncpdc[state][attribute].index(value)
       return cpdc[state][attribute][index]
```

```
   from math import sqrt

   def euclideanDist(x,y):
     distance = sqrt((x-y)**2)
     return distance
```

```
[ ]  def solution(pState,flag):
       if flag.count(1) == 0:
         print("All data entered is Consistent for the confirmed diagnosis.")
       else:
         print(flag.count(1)," data entered is Inconsistent for the confirmed diagnosis.")

       print("The complete and most appropriate values of attributes for the confirmed diagnosis are:\n")
       printDiagnosis(pState)
       print("Patient is in ",printState(pState)," state of pediatric asthma.")
```

```
[ ]  def printDiagnosis(pState):
       attributes = domain.keys()
       for attribute in attributes:
         print(attribute," : ",pState.att[attribute])
```

```
[ ]  def printState(pState):
       if pState.state == "DDS1":
         return "mild"
       elif pState.state == "DDS2":
         return "moderate"
       elif pState.state == "DDS3":
         return "severe"
```

DEMONSTRATION

FIn the following code we present the case where incomplete health attribute values and a diagnosis of mild asthma are fed inside the program and it return the complete CPG of the case. The doctor may use this document to suggest a possible treatment plan for the patient.

Attribute	Value
Dyspnea	exertional
Beta Agonist	good response
Difficult Speech	absent
Tachycardia	absent
Typical Episode	worse

```
printDiagnosis(pState)
check1(pState)
```

```
dyspnea  :  exertional
beta agonist  :  good response
difficult speech  :  absent
tachycardia  :  absent
tachypnea  :  NO
accessory muscles  :  NO
breathing sounds  :  NO
typical episode  :  worse
sao2  :  NO
pev  :  NO
fev  :  NO
All data entered is Consistent for the confirmed diagnosis.
The complete and most appropriate values of attributes for the confirmed diagnosis are:

dyspnea  :  exertional
beta agonist  :  good response
difficult speech  :  absent
tachycardia  :  absent
tachypnea  :  exertional
accessory muscles  :  none
breathing sounds  :  normal
typical episode  :  worse
sao2  :  96
pev  :  76
fev  :  76
Patient is in  mild  state of pediatric asthma.
```

FURTHER OPTIMIZATION

We have executed the current CPG project using a few constraints and it has a lot more constraint options that can be applied to it.

- For incomplete health attributes, we can assign values using Euclidian distance if a relation between different heath attributes(or their weightage for different DDS is found) are found.
- We can further add treatment plans to facilitate doctors in the given CPG.
- We can add more constraints/define hueristics to the procedure based upon details received from the more details received by doctors to make the process more accurate and fast.
- We can add machine learning methods to the process which will learn from earlier predictions of CPG and make the process more efficient.

CONCLUSION

In this project, we have introduced a data-driven CPG constraint satisfaction approach for implementing CPG in clinical practice. By proposing a method for implementing CPG in cases of incomplete data or a mismatch between patient data and the CPG, the approach expands the decision step of guideline models.

Using constraint satisfaction, incomplete or mismatched data can be extended into complete solutions for describing a patient's condition. We also provide a methodology for a data-driven explanation of the differing effects of mismatched clinical data on the final description of a patient state using constraint satisfaction to leverage different patient 'scenarios' which can include/exclude mismatched data as well as allowing physicians to view the extent to which they are applying a guideline for a particular patient. By highlighting mismatched data and ensuring the collection of all data for each patient encounter, the framework also improves data quality.