علیرضا کمرزردی و امین امیری

داکیومنت پروژه ارضای محدودیت درس هوش مصنوعی

**فاز یک**

هر سالن در مسأله متناظر با یک آبجکت از کلاس است که دارای دامنه و محدودیت هاست.

class Hall:  
 def \_\_init\_\_(*self*, hall: **'Hall'** = None):  
 if hall is None:  
 *self*.domain = set()  
 *self*.constraint = set()  
 *self*.parent = set()

هر شی از کلاس از متغیر مسأله نگهداری میکند. برای مثال لیست تمام سالن ها و تمام یال هارا در اینجا ذخیره میکنیم.

class CSP:  
 def \_\_init\_\_(*self*, csp: **'CSP'** = None) -> None:  
 if csp is None:  
 *self*.n = None  
 *self*.m = None  
 *self*.e = None  
 *self*.halls = {}  
 *self*.constraints = set()

**فاز دو**

ما در تابع ابتکاری به دنبال مقدار هایی از دامنه هستیم که کمترین تداخل را با مقدار های دامنه نود های همسایه دارد. در واقع این تابع لیست مرتب شده از مقادیر دامنه بر اساس کمترین تداخل بصورت صعودی به ما میدهد.

def lcv(hall: Hall, assignment) -> list:  
 return sorted(hall.domain, key=lambda value: number\_of\_conflicts(hall, value, assignment))

تابع تعداد تداخل های سالن با مقدار با سایر همسایگان مقدار داده شده اش را میدهد.

* **کمترین مقادیر باقی ماننده ( MRV )**

این از بین متغیر های موجود مغیری را برای مقداردهی انتخاب میکند که کمترین را دارد .

طبق مدل در نظر گرفته شده از بین های موجود را انتخاب میکنیم که متقاضی کمتری ( دامنه کوچک ) داشته باشد .

برای این کار تمام ها را بر اساس تعداد دامنه ها در لیستی به صورت صعودی مرتب میکنیم سپس اولین که مقدار مقداردهی نشده است را انتخاب میکنیم .

فرض کنیم تعداد ها باشد ، در این صورت در نظر گرفتن تعداد دامنه ها در یک لیست و مرتب کردن آن

می­باشد و انتخاب با کمترین دامنه در بدترین حالت میباشد .

*function MRV( csp , assignment ) return hall minimum remaining values*

*local varibles: list , a halls domain list*

*sort(list)*

*for each hall in list do*

*if hall not assign then*

*return hall*

* **کنترل روبه جلو ( forward checking )**

این هنگامی که یک متغیر مقدار دهی می­شود ، متغیر هایی که دامنه آن ها با مقداردهی این متغیر محدود می­شود را محدود میکند . اگر با مقدار دهی یک متغیر و محدود کردن دامنه های تاثیرپذیر آن ، متغیر وجود داشت که مقدار دهی نشده بود و دامنه اش بدون عضو بود جستجو متوقف شود . طبق مدل در متغیر لیست اندیس هایی است می­توان به آن رفت ( همسایه های خروجی ) و لیست اندیس هایی است که از آن می­توان به این آمد. ( همسایه های ورودی ) با محدود کردن دامنه های این دو مجموعه میتوان آن را پیاده سازی کرد . در بد ترین حالت هر با تمام ها همسایه است پس مرتبه آن از است که تعداد ها می­باشد .

*function forward-checking( csp , hall, value ) return hall minimum remaining values*

*local varibles: list , a halls list union constraint and parent hall*

*for each hall in list do*

*if value in hall domain then*

*remove value from hall*

* **پس گرد ( backtracking )**

الگوریتم پس گرد ، الگوریتم جستجوی ساختار یافته است که با استفاده از یک درخت فضای حالت همه راه حل های ممکن را می­یابد .

این روش برای حل مسائل به صورت بازگشتی به کار برده می­شود و به تمام راه حل های یک مسئله دست پیدا میکند .

در روش عقب گرد مسئله دارای محدودیت هایی است و راه حل هایی که به جواب نمی­رسند ادامه پیدا نمی­کنند و حذف میشوند .

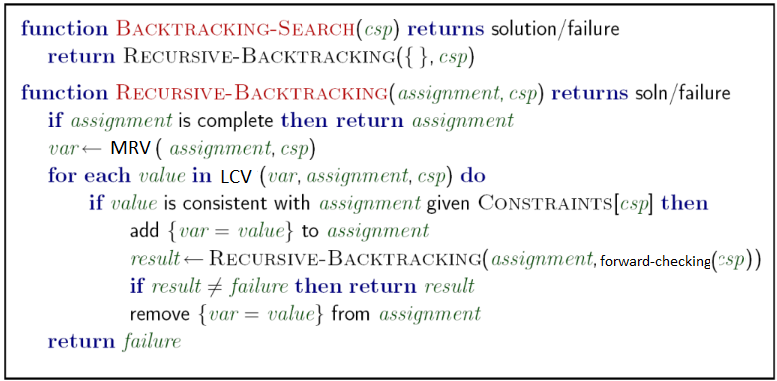
در این مدل دارای این محدودیت است که هر از بعدی خارج می­شود ( یا اگر بعدی نداشت خودش ) و بعدی نباید

مقدار یکسانی با کنونی داشته باشد .

پیاده سازی الگوریتم پس گرد در این پروژه همراه با های ، و می­باشد .

به این صورت که هر عمق توسط و ترتیب دامنه های انتخاب شده برای و درخت از چپ به راست

توسط و محدود کردن دامنه بعد از مقدار دهی توسط صورت میگیرد .



**فاز سه**

برای اینکه هر متغیری دارای سازگاری یال باشد، الگوریتم یال هایی که باید در نظر گرفته شوند، در یک صف نگهداری میکند.در آغاز، صف شامل تمام تمام یال های موجود در است. سپس یال را از صف خارج و کاری میکند که نسبت به دارای سازگاری یال باشد. اگر با این کار csp.halls[i].domain طوری تغییر کند که فاقد عضو باشد، انگاه پی میبریم که فاقد جواب سازگار است. در غیر این صورت همه یال هایرا به صف اضافه میکنیم که همسایه است. علت این کار این است که تغییر در دامنه سالن i ممکن است باعث کاهش بیشتری در دامنه سالن k شود. حتی اگر آن را قبلا در نظر گرفته ایم.

برسی را ادامه میدهیم و سعس میکنیم مقادیر را از دامنه متغیر ها حذف کنیم تا هیچ یالی در صف باقی نماند. در این نقطه به ای رسیدیم که معادل با اصلی است. اما باسازگاری یال اغلب باعث میشود جست و جو سریع تر انجام شود.

def ac3(csp: CSP) -> bool:  
 queue = list(csp.constraints)  
  
 while len(queue) != 0:  
 i, j = queue.pop(0)  
 if revise(csp, i, j):  
 if len(csp.halls[i].domain) == 0:  
 return False  
 for k in (csp.halls[i].constraint - {j}):  
 if (k, i) not in queue:  
 queue.append((k, i))  
  
 return True  
  
  
def revise(csp, i, j) -> bool:  
 revised = False  
  
 for value in csp.halls[i].domain.copy():  
 if len(csp.halls[j].domain - {value}) == 0:  
 csp.halls[i].domain.remove(value)  
 revised = True  
  
 return revised

پیچیدگی زمانی یک با متغیر که اندازه دامنه هر کدام است، است.