علیرضا کمرزردی و امین امیری

داکیومنت پروژه ارضای محدودیت درس هوش مصنوعی

**فاز یک**

* **Hall**
* **CSP**

**فاز دو**

* **LCV**

ما در تابع ابتکاری lcv 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))

* **MRV**
* **Forward\_checking**
* **backtracking**

**فاز سه**

* **AC3**

برای اینکه هر متغیری دارای سازگاری یال باشد، الگوریتم AC3 یال هایی که باید در نظر گرفته شوند، در یک صف نگهداری میکند.در آغاز، صف شامل تمام تمام یال های موجود در CSP است. سپس AC3 یال (i, j) را از صف خارج و کاری میکند که i نسبت به j دارای سازگاری یال باشد. اگر با این کار csp.halls[i].domain طوری تغییر کند که فاقد عضو باشد، انگاه پی میبریم که csp فاقد جواب سازگار است. در غیر این صورت همه یال های (k, i) را به صف اضافه میکنیم که k همسایه i 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 csp ای رسیدیم که معادل با csp اصلی است. اما csp 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 AC3 یک csp با n متغیر که اندازه دامنه هر کدام d است، است.