

# داکیومنت پروژه ارضای محدودیت درس هوش مصنوعی

علیرضا کمزردی و امین امیری

## فاز یک

### • Hall

هر سالن در مسأله متناظر با یک آبجکت از کلاس *Hall* 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()
```

### • CSP

هر شی از کلاس *CSP* 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

### • 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))
```

تابع *number\_of\_conflicts* تعداد تداخل های سالن با مقدار *value* با سایر همسایگان مقدار داده شده اش را میدهد.

### • کمترین مقادیر باقی مانده ( MRV )

این *heuristic* از بین متغیر های موجود مغیری را برای مقداردهی انتخاب میکند که کمترین *Domain* را دارد .  
طبق مدل در نظر گرفته شده از بین *hall* های موجود *hall* را انتخاب میکنیم که متقاضی کمتری ( دامنه کوچک ) داشته باشد .  
برای این کار تمام *hall* ها را بر اساس تعداد دامنه ها در لیستی به صورت صعودی مرتب میکنیم سپس اولین *hall* که مقدار مقداردهی نشده است را انتخاب میکنیم .  
فرض کنیم تعداد *hall* ها  $n$  باشد ، در این صورت در نظر گرفتن تعداد دامنه ها در یک لیست  $O(n)$  و مرتب کردن آن  $O(n \log n)$  می باشد و انتخاب *hall* با کمترین دامنه در بدترین حالت  $O(n)$  میباشد .

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

```
    local variables: 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 *heuristic* 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 *hall* متغیر *constraint* لیست اندیس *hall* هایی است می‌توان به آن رفت ( همسایه های خروجی ) و *parent* لیست اندیس *hall* هایی است که از آن می‌توان به این *hall* آمد . ( همسایه های ورودی ) با محدود کردن دامنه های این دو مجموعه *hall* میتوان آن را پیاده سازی کرد . در بدترین حالت هر *hall* با تمام *hall* ها همسایه است پس مرتبه آن از  $O(n)$  است که  $n$  تعداد *hall* ها می‌باشد .

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

```
local variables: 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 می یابد .  
این روش برای حل مسائل به صورت بازگشتی به کار برده می شود 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 نمی رسند ادامه پیدا نمی کنند و حذف میشوند .  
در این مدل *hall* دارای این محدودیت است که هر *hall* از *hall* بعدی خارج می شود ( یا اگر بعدی نداشت خودش ) و *hall* بعدی نباید مقدار یکسانی با *hall* کنونی داشته باشد .  
پایاده سازی الگوریتم پس گرد در این پروژه همراه با *heuristic* های *forward checking* ، *lcv* و *mrsv* می باشد .  
به این صورت که *hall* هر عمق توسط *mrsv* و ترتیب دامنه های *hall* انتخاب شده برای *assign* و *expand* درخت از چپ به راست توسط *lcv* و محدود کردن دامنه بعد از مقدار دهی توسط *forward chekcing* صورت میگیرد .

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING( { }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← MRV ( assignment, csp)
  for each value in LCV ( var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add { var = value } to assignment
      result ← RECURSIVE-BACKTRACKING( assignment, forward-checking(csp) )
      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

## 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 سپس 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  $csp$  با سازگاری 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،  $O(cd^3)$  است.