**Комп’ютерний практикум №5**

**Моделі на основі змінних в задачах задоволення обмежень**

**ПІБ:** Шляхтун Денис Михайлович.

**Група:** ІП-14.

**Мета роботи:** ознайомитись з методами пошуку рішень в моделях ШІ на основі змінних.

**Завдання:** розв’язати задачу задоволення обмежень згідно з варіантом в обраному середовищі, реалізувавши запропоновані методи пошуку. Порівняти реалізації між собою та з базовим алгоритмом пошуку з поверненням. Виконати міні-дослідження впливу параметру задачі. Підготувати звіт

**Номер варіанту:** 22.

**Завдання для варіанту:** ферзі, BT+MCV+LCV+AC-3, BT+LCV.

**Задача:** задача з ферзями: потрібно розставити на шаховій дошці ферзів так, що жоден з них не ставить під удар один одного. Тобто, вони не повинні стояти в одній вертикалі, горизонталі чи діагоналі. Кількість ферзів відповідає розмірності квадратної шахової дошки, наприклад для дошки 8х8 потрібно розставити 8 ферзів.

**Середовище:** шахова дошка заданого розміру (8х8, 12х12, 16х16, 20x20).

**Методи вирішення задачі:**

BT – метод пошуку з поверненням, працює з частковими присвоюваннями, детермінований – завжди працює за одними і тими ж правилами і не має випадковостей, глобальний – досліджує всі можливі варіанти, оптимальний – через глобальність можливо знайти оптимальний варіант.

MCV – (most constrained variable) – евристика, що забезпечує динамічне упорядкування змінних при їх виборі: спочатку обираються найбільш обмежені, тобто ті, що мають менше можливих значень.

LCV – (least constrained value) – евристика, що забезпечує динамічне упорядкування значень змінних при їх виборі: спочатку обираються найменш обмежені, тобто ті, що мають менше можливих значень.

AC-3 – LOOKAHEAD евристика перевірки узгодженості всіх дуг (обмежень), для зменшення допустимих значень всіх змінних.

**Реалізація методу:**

def get\_unconstrained\_values\_count(self, assignment: Dict[V, D] = {}, variable: V = None) -> int:

count = 0

for value in self.domains[variable]:

local\_assignment = assignment.copy()

local\_assignment[variable] = value

if self.consistent(variable, local\_assignment):

count += 1

return count

def MCV(self, assignment: Dict[V, D] = {}, unassigned: List[V] = {}) -> V:

result = [self.get\_unconstrained\_values\_count(assignment, variable) for variable in unassigned]

return unassigned[result.index(min(result))]

def LCV(self, assignment: Dict[V, D] = {}, variable: V = None) -> List[D]:

values = []

result = []

unassigned: List[V] = [v for v in self.variables if v not in assignment and v != variable]

for value in self.domains[variable]:

local\_assignment = assignment.copy()

local\_assignment[variable] = value

if self.consistent(variable, local\_assignment):

values.append(value)

count = 0

for unassign in unassigned:

count += self.get\_unconstrained\_values\_count(local\_assignment, unassign)

result.append(count)

return [element for \_, element in sorted(zip(result, values), key=lambda x: x[0], reverse=True)]

def AC\_3(self, assignment: Dict[V, D] = {}) -> bool:

unassigned: List[V] = [v for v in self.variables if v not in assignment]

for variable in unassigned:

if self.get\_unconstrained\_values\_count(assignment, variable) == 0:

return False

return True

# BT+MCV+LCV+AC-3

def backtracking\_search\_method\_1(self, assignment: Dict[V, D] = {}) -> Optional[Dict[V, D]]:

self.BT\_calls += 1

# assignment is complete if every variable is assigned (our base case)

if len(assignment) == len(self.variables):

return assignment

# get all variables in the CSP but not in the assignment

unassigned: List[V] = [v for v in self.variables if v not in assignment]

# get the every possible domain value of the most constrained value

first: V = self.MCV(assignment, unassigned)

for value in self.LCV(assignment, first):

local\_assignment = assignment.copy()

local\_assignment[first] = value

# if we're still consistent, we recurse (continue)

if self.AC\_3(local\_assignment):

result: Optional[Dict[V, D]] = self.backtracking\_search\_method\_1(local\_assignment)

# if we didn't find the result, we will end up backtracking

if result is not None:

return result

return None

**Результати застосування розробленого методу:** було обрано метрику кількості викликів функції пошуку з поверненням, час роботи міг бути гарним показником, але для більшості розмірів дошки програма виконується менше 2 секунд і похибка може зробити результати вимірів нерепрезентативними.

Результат роботи програми:

Зображення, що містить текст, знімок екрана, Шрифт, чорний

Автоматично згенерований опис

**Оцінка результатів:** загалом краще всіх впоралася модифікація алгоритму BT+MCV+LCV+AC-3, у 3 з 4 запусків експерименту функція пошуку з поверненням викликається у межах 26-30 разів, тобто немає сильної залежності від розміру дошки, ніж при відсутності евристик. Використання лише однієї евристики LCV теж показує значено кращі результати за звичайний метод, але в середньому гірші за більшу кількість евристик. З іншої сторони, в цьому випадку було б доцільно перевірити час виконання, адже для обрахунку евристик потрібен час, але кількість викликів функції доволі незначна, щоб можна було провести точний аналіз швидкодії.

**Міні-дослідження:**

|  |  |  |  |
| --- | --- | --- | --- |
| Розмір дошки | BT | BT+MCV+LCV+AC-3 | BT+LCV |
| 8x8 | 114 | 30 | 56 |
| 12x12 | 262 | 26 | 259 |
| 16x16 | 10053 | 240 | 2725 |
| 20x20 | 199636 | 28 | 94 |

У методі без використання евристик помітна пряма залежність кількості викликів функції від розміру дошки. При використанні евристик така залежність не прослідковується, тому при збільшенні розміру дошки евристики значно краще впоралися із задачею у порівнянні з їхньою відсутністю.