مستندات پروژه: پیاده‌سازی حل‌کننده بازی فوتوشیکی با CSP استفاده از

۱. مقدمه

می‌پردازد. بازی فوتوشیکی یک پازل (Futoshiki) این پروژه به پیاده‌سازی یک حل‌کننده هوشمند برای بازی فوتوشیکی پر n انجام می‌شود. هدف این است که خانه‌های جدول با اعداد ۱ تا (n×n) منطقی عددی است که در یک شبکه مربعی شوند، به طوری که هر عدد فقط یک بار در هر سطر و ستون ظاهر شود و همچنین محدودیت‌های نابرابری (کوچکتر/بزرگتر) بین خانه‌های مجاور رعایت گردند. این مسئله به خوبی می‌تواند به عنوان یک مسئله ارضای محدودیت‌ها (Constraint Satisfaction Problem - CSP) مدل‌سازی و حل شود.

۲. مدل‌سازی فوتوشیکی به عنوان CSP

برای مدل‌سازی بازی فوتوشیکی به عنوان یک CSP تعریف می‌شوند:، عناصر زیر

- **متغیرها (Variables):** هر خانه از جدول فوتوشیکی یک متغیر است. هر متغیر با یک تاپل (r,c) نشان داده می‌شود که r شماره سطر و c شماره ستون است.
- **دامنه‌ها (Domains):** دامنه هر متغیر، مجموعه‌ای از اعداد ممکن است که می‌تواند در آن خانه قرار گیرد. برای یک شبکه ntimesn، دامنه اولیه هر خانه `{1, 2, ..., n}` است. اگر خانه‌ای در ابتدا دارای مقدار ثابت باشد، دامنه آن تنها شامل همان مقدار ثابت خواهد بود.
- **محدودیت‌ها (Constraints):**
  - **محدودیت‌های `All-Different`:**
    - **محدودیت‌های سطری:** در هر سطر، تمام خانه‌ها باید مقادیر متمایزی داشته باشند.
    - **محدودیت‌های ستونی:** در هر ستون، تمام خانه‌ها باید مقادیر متمایزی داشته باشند.
  - **محدودیت‌های نابرابری (Inequality Constraints):** بین برخی خانه‌های مجاور، محدودیت‌های وجود دارد که باید رعایت شوند. این محدودیت‌ها به صورت () "یا "بزرگتر از ($\<$) ""کوچکتر از سه تایی `((r1, c1), (r2, c2), op)` تعریف می‌شوند که op می‌تواند > یا < باشد.

۳. الگوریتم‌ها و تکنیک‌های پیاده‌سازی

به عنوان پایه استفاده می‌کند و سپس آن را با (Backtracking Search) این پروژه از الگوریتم جستجوی عقب‌گرد تکنیک‌های پیشرفته‌تر CSP برای بهبود کارایی تقویت می‌کند.

۳/۱. جستجوی عقب‌گرد (Backtracking Search)

است که به صورت بازگشتی، متغیرها را یکی یکی (Depth-First Search) این الگوریتم یک جستجوی عمق-اول مقداردهی می‌کند. اگر یک مقداردهی منجر به نقض محدودیت شود، الگوریتم به عقب برگشته و مقدار دیگری را امتحان می‌کند.

۳/۲. تکنیک‌های استنتاج (Inference Techniques)

برای کاهش فضای جستجو و کشف سریع‌تر ناسازگاری‌ها:

- **بررسی رو به جلو (Forward Checking - FC):**
  - دامنه‌های متغیرهای همسایه Forward Checking، (X = v) پس از هر مقداردهی به یک متغیر (Y) را بررسی می‌کند.
  - اگر دامنه هر متغیر همسایه پس از این حذف‌ها خالی شود، نشان‌دهنده ناسازگاری است و الگوریتم بلافاصله عقب‌گرد می‌کند.
  - **کاربرد:** Forward Checking در این پروژه، بلافاصله پس از مقداردهی به یک متغیر و قبل از ادامه جستجو فراخوانی می‌شود CSPSolver.
- **یکنواختی قوس‌ها (Arc Consistency - AC-2):**
  - این تکنیک اطمینان حاصل می‌کند که برای هر جفت متغیر (Xi, Xj) که با هم محدودیت دارند، و وجود دارد که جفت (x, y) در دامنه D(Xj)، حداقل یک مقدار y در دامنه D(Xi) برای هر مقدار x وجود دارد. اگر چنین xای یافت نشود، y را ارضا کند. از D(Xi) حذف می‌شود xj و Xi محدودیت بین.
  - این فرآیند به صورت تکراری بر روی یک صف از قوس‌ها (Constraints) اعمال می‌شود تا زمانی که هیچ دامنه‌ای دیگر تغییر نکند یا یک دامنه خالی شود.
  - **کاربرد:**
    - **پیش‌پردازش (Preprocessing):** AC-2 می‌تواند یک بار در ابتدای حل، قبل از شروع جستجوی عقب‌گرد، اجرا شود تا دامنه‌ها را تا حد امکان کوچک کند.
    - **انتشار محدودیت‌ها (Constraint Propagation):** پس از هر مقداردهی موفقیت‌آمیز به یک متغیر، AC-2 می‌تواند دوباره اجرا شود تا تأثیر این مقداردهی بر روی سایر دامنه‌ها انتشار یابد و ناسازگاری‌ها زودتر شناسایی شوند.

۳/۳. هیوریستیک‌های انتخاب متغیر و مقدار (Variable and Value Ordering Heuristics)

این هیوریستیک‌ها برای بهبود کارایی جستجو استفاده می‌شوند:

- **Minimum Remaining Values (MRV - متغیر با کمترین مقادیر باقیمانده):**
  - در هر مرحله از جستجو، متغیری انتخاب می‌شود که کوچکترین دامنه (کمترین تعداد مقادیر ممکن) را دارد.
  - **منطق:** این هیوریستیک به دنبال "سخت‌ترین" متغیر است. با حل کردن متغیرهای سخت‌تر در ابتدا، احتمال کشف بن‌بست‌ها در مراحل اولیه جستجو افزایش می‌یابد و از هدر رفتن زمان در شاخه‌های بی‌نتیجه جلوگیری می‌شود.
- **Least Constraining Value (LCV - کمترین مقدار محدودکننده):**
  - هنگامی که یک متغیر برای مقداردهی انتخاب شد، مقادیری از دامنه آن به ترتیب انتخاب می‌شوند که کمترین محدودیت را بر دامنه‌های متغیرهای همسایه اعمال می‌کنند.
  - **منطق:** این هیوریستیک سعی می‌کند "آسان‌ترین" مقدار را انتخاب کند. با انتخاب مقادیری که بیشترین گزینه‌ها را برای متغیرهای آینده باقی می‌گذارند، احتمال اینکه به بن‌بست برسیم و نیاز به عقب‌گرد داشته باشیم، کاهش می‌یابد.

۴. ساختار پروژه

پروژه به چندین فایل پایتون تقسیم شده است که هر یک مسئولیت‌های خاصی را بر عهده دارند:

- **futoshiki_board.py:**
  - کلاس FutoshikiBoard که نمایانگر وضعیت فعلی جدول فوتوشیکی است.

- مسئول نگهداری `grid` (خانه‌های جدول)، `domains` (دامنه‌های هر خانه) و `inequality_constraints`.
- پیاده‌سازی متدهایی برای `assign` (مقداردهی)، `unassign` (برگرداندن مقداردهی)، `get_unassigned_variables` (گرفتن متغیرهای خالی)، `is_consistent` (بررسی سازگاری یک مقداردهی)، و `get_neighbors` (گرفتن همسایه‌ها).
- متد `display` برای نمایش بصری جدول.

- **`heuristics.py`:**
  - پیاده‌سازی تابع `select_unassigned_variable_mrv` برای انتخاب متغیر با استفاده از MRV.
  - پیاده‌سازی تابع `order_domain_values_lcv` برای مرتب‌سازی مقادیر دامنه با استفاده از LCV.
- **`inference.py`:**
  - پیاده‌سازی تابع `forward_check` برای اجرای Forward Checking.
  - پیاده‌سازی تابع `revise` که یک مرحله اصلی در AC-2 است.
  - پیاده‌سازی تابع `ac2` برای اجرای الگوریتم Arc Consistency.
- **`csp_solver.py`:**
  - کلاس `CSPSolver` که موتور اصلی حل‌کننده CSP است.
  - پیاده‌سازی متد `solve` که نقطه شروع حل است.
  - پیاده‌سازی متد `backtracking_search` که هسته الگوریتم را تشکیل می‌دهد.
  - منطق ادغام `Forward Checking`، `AC-2`، `MRV` و `LCV` در فرآیند جستجو.
  - شمارش تعداد `backtracks` و زمان اجرا.
- **`futoshiki_solver.py`:**
  - فایل اصلی و نقطه ورود برنامه.
  - حاوی تعریف پازل‌های فوتوشیکی به صورت `hardcoded` (بدون نیاز به ورودی کاربر).
  - فراخوانی `CSPSolver` ساده (فقط backtracking) در دو حالت "بهینه" (با همه تکنیک‌ها) و "بهینه".
  - نمایش خروجی‌های متنی شامل جدول‌های مقایسه زمان و تعداد برگشت‌ها.
  - برای مقایسه بصری زمان و تعداد بازگشت‌ها با استفاده از `matplotlib` **افزودن قابلیت رسم نمودار**.

۵. نحوه اجرا

برای اجرای پروژه:

1. **نصب پیش‌نیازها:**

   Bash

   ```
   pip install matplotlib
   ```

2. **ذخیره فایل‌ها:** تمام فایل‌های پایتون (`futoshiki_board.py`, `heuristics.py`, `inference.py`, `csp_solver.py`, `futoshiki_solver.py`) را در یک پوشه قرار دهید.
3. **اجرا:** از طریق ترمینال یا خط فرمان، به پوشه پروژه رفته و دستور زیر را اجرا کنید:

   Bash

   ```
   python futoshiki_solver.py
   ```

را حل کرده و نتایج را در کنسول و به `futoshiki_solver.py`برنامه به صورت خودکار پازل‌های تعریف شده در صورت نمودار نمایش خواهد داد.

۶. خروجی پروژه

خروجی پروژه شامل موارد زیر است:

- **حل کامل جدول:** برای هر پازل، وضعیت اولیه و در صورت یافتن راه‌حل، جدول حل شده نمایش داده می‌شود.
- **گزارش عملکرد در کنسول:**
  - زمان سپری شده برای هر دو نسخه حل‌کننده (ساده و بهینه).
  - برای هر دو نسخه (`Backtrack Count`) تعداد برگشت‌ها.
  - یک جدول مقایسه‌ای خلاصه از زمان و تعداد برگشت‌ها برای هر پازل.
  - نتیجه‌گیری متنی در مورد عملکرد بهتر حل‌کننده بهینه (معمولاً سریع‌تر با برگشت‌های کمتر).
- **نمودارهای مقایسه‌ای:**
  - برای هر پازل، دو نمودار میله‌ای (Bar Chart) جداگانه نمایش داده می‌شود:
    - نمودار مقایسه زمان حل (بر حسب ثانیه).
    - نمودار مقایسه تعداد برگشت‌ها.
  - این نمودارها به صورت بصری برتری حل‌کننده بهینه را در کاهش زمان و تعداد جستجو نشان می‌دهند.