۹۹۵۲۱۴۶۹ سینا علی‌نژاد — تمرین هوش مصنوعی HW4

Year: Month: Date: ( ) Subject

تئوری:

سوال ۱-

الف) تولید جمعیت اولیه:

نحوهٔ encoding: برای هر رنگ یک شماره از ۱ تا n در نظر می‌گیریم که n تعداد رنگ‌هاست. برای مثال به ازای n=۴ داریم:

(۱,۱,۳,۲,۲,۴,۳,۴,۱,۲,۱,۱,۴,۳,۳)

رنگ شهر ۰ — ۱ — رنگ شهر ۱۴.

یک کران بالای خوب برای n، یکی بیشتر از ماکزیمم درجهٔ رئوس است که در این مثال می‌شود n=۸

تولید جمعیت اولیه: (برای مثال به ازای n = ۸) جمعیت ۶

C_1 = (۱,۱,۲,۴,۱,۳,۸,۸,۷,۵,۷,۶,۱,۴,۶)

C_2 = (۴,۸,۳,۳,۱,۷,۲,۴,۸,۵,۵,۶,۷,۸,۶)

C_3 = (۲,۸,۷,۱,۱,۴,۳,۴,۴,۵,۲,۶,۳,۴,۵)

C_4 = (۷,۵,۴,۲,۲,۴,۴,۲,۴,۵,۴,۸,۶,۷,۳)

C_5 = (۱,۱,۲,۱,۴,۱,۷,۵,۱,۷,۷,۲,۸,۲,۱)

C_6 = (۳,۳,۵,۷,۶,۱,۴,۵,۵,۷,۷,۳,۸,۸,۵)

اعداد بالا توسط برنامه به صورت رندوم تولید شدند.

ب) محاسبه Fitness : هر یال دو رأس را به یکدیگر وصل می‌کند.

مجموع تعداد یال‌هایی که دو رأس غیرهمرنگ را به هم وصل می‌کنند را به عنوان fitness value می‌گیریم. هرچه این عدد برای یک کروموزوم بیشتر باشد، بهتر است.

$f(C_۱) = ۱۸$, $f(C_۲) = ۱۸$, $f(C_۳) = ۲۱$, $f(C_۴) = ۲۱$
$f(C_۵) = ۱۵$, $f(C_۶) = ۲۰$

اعداد بالا با برنامه بدست آمدند.

به ترتیب fit بودن $C_۴, C_۳, C_۶, C_۱, C_۲, C_۵$

ج) Cross over , Mutation :

اگر بخواهیم دقیق‌تر عمل کرده باشیم، می‌توانیم به نسبت عدد fit به هر کروموزوم یک درصدی اختصاص دهیم که با این احتمال آن کروموزوم انتخاب شود ولی اینجا می‌توانیم از بالا به پایین به ترتیب fit بودن اقدام به Crossover روی زوج کروموزوم‌ها کنیم.

برای Cross over یک قطعه‌ای از کروموزوم را انتخاب کرده و یک نیمهٔ چپ را از یک کروموزوم و نیمه راست را از کروموزوم دیگر می‌گیریم.

$C_۴, C_۳$: (۷, ۵, ۴, ۲ | ۳, ۴, ۴, ۲, ۴, ۵, ۴, ۸, ۶, ۷, ۳)
(۲, ۸, ۷, ۱ | ۱, ۴, ۳, ۴, ۴, ۵, ۲, ۶, ۳, ۴, ۵)

Crossover : (۷, ۵, ۴, ۲, ۱, ۴, ۳, ۴, ۴, ۵, ۲, ۶, ۳, ۴, ۵) $C'_۱$
(۲, ۸, ۷, ۱, ۳, ۴, ۴, ۲, ۴, ۵, ۴, ۸, ۶, ۷, ۳) $C'_۲$

Year: Month: Date: ( ) Subject

C۶ , C۱ : (۳ , ۳ , ۵ , ۷ } ۶ , ۱ , ۴ , ۵ , ۵ , ۷ , ۷ , ۳ , ۸ , ۸ , ۵)
(۱ , ۱ , ۲ , ۴ { ۱ , ۳ , ۸ , ۸ , ۷ , ۵ , ۷ , ۶ , ۱ , ۴ , ۶)

crossover : (۳ , ۳ , ۵ , ۷ , ۱ , ۳ , ۸ , ۸ , ۷ , ۵ , ۷ , ۶ , ۱ , ۴ , ۶) C'۳
(۱ , ۱ , ۲ , ۴ , ۶ , ۱ , ۴ , ۵ , ۵ , ۷ , ۷ , ۳ , ۸ , ۸ , ۵) C'۴

C۲ , C۵ : (۴ , ۸ , ۳ , ۳ } ۱ , ۷ , ۲ , ۴ , ۸ , ۵ , ۵ , ۶ , ۷ , ۸ , ۶)
(۱ , ۱ , ۲ , ۱ { ۴ , ۱ , ۷ , ۵ , ۱ , ۷ , ۷ , ۲ , ۸ , ۲ , ۱)

crossover : (۴ , ۸ , ۳ , ۳ , ۴ , ۱ , ۷ , ۵ , ۱ , ۷ , ۷ , ۲ , ۸ , ۲ , ۱) C'۵
(۱ , ۱ , ۲ , ۱ , ۱ , ۷ , ۲ , ۴ , ۸ , ۵ , ۵ , ۶ , ۷ , ۸ , ۶) C'۶

mutation : برای هر کروموزوم به احتمال کمی (مثلاً ۱۰ درصد)
ممکن است mutation یا جهش ژنی رخ دهد و یک یا چند تا از
اعداد آن به صورت رندوم عوض شوند. اعدادی که با رنگ آبی هستند دچار
جهش شده اند.

C'۱ : (۷ , ۵ , ۴ , ۲ , ۱ , ۴ , ۳ , ۴ , ۶ , ۵ , ۲ , ۶ , ۳ , ۴ , ۵)

C'۲ : (۲ , ۸ , ۷ , ۱ , ۳ , ۴ , ۴ , ۲ , ۴ , ۵ , ۴ , ۸ , ۶ , ۷ , ۳)

C'۳ : (۳ , ۳ , ۵ , ۷ , ۴ , ۳ , ۸ , ۸ , ۷ , ۵ , ۷ , ۶ , ۱ , ۴ , ۶)

C'۴ : (۱ , ۱ , ۲ , ۴ , ۶ , ۱ , ۴ , ۵ , ۵ , ۷ , ۷ , ۳ , ۸ , ۱ , ۵)

C'۵ : (۴ , ۸ , ۳ , ۳ , ۴ , ۱ , ۷ , ۵ , ۱ , ۷ , ۷ , ۲ , ۸ , ۲ , ۱)

C'۶ : (۱ , ۱ , ۲ , ۱ , ۱ , ۷ , ۲ , ۴ , ۸ , ۵ , ۵ , ۶ , ۷ , ۸ , ۶)

د) نسل جدید :

$f(c'_{۱}) = ۲۰$ $f(c'_{۲}) = ۲۰$ $f(c'_{۳}) = ۱۷$

$f(c'_{۴}) = ۱۷$ $f(c'_{۵}) = ۱۸$ $f(c'_{۶}) = ۱۷$

جمع fitness نسل جدید : ۱۰۹

" " نسل قبلی : ۱۱۴

در نسل جدید مجموع fitness مقدار کمتری نسبت به نسل قبلی دارد
پس در اینجا پسرفت داشتیم ، ولی با طی چندین مرحله به یک
local optimum همگرا می‌شویم که با احتمال خوبی می‌تواند global optimum باشد ،
که این احتمال به جمعیت اولیه هم بستگی دارد . البته در اینجا بین نسل اولیه
جواب وجود داشت ، برای همین تاثیر آن مشخص نشد .

گزارش کد

Year: Month: Date: ( ) Subject

توابع کلاس NQueens :

تابع initial : یک تاپل برمی‌گرداند که در واقع استیت اولیه صفحه شطرنج است و به صورت رندوم تولید می‌شود. برای مثال هر ، متغیر ماست ، که مقداری که می‌گیرد ، شماره ستونی است که وزیرِ ردیف در آن قرار دارد.

تابع goal_test : این تابع یک استیت به عنوان ورودی می‌گیرد و برای هر جفت متغیر بررسی می‌کند که دو وزیر همدیگر را نزنند ، اگر حداقل یک جفت باشند که در این وضعیت باشند ، False برمی‌گردانیم ، در غیر اینصورت True .

تابع threatening_states : این تابع بررسی می‌کند که دو موقعیت که وزیر در آن‌ها است ، آیا همدیگر را می‌زنند یا نه ، بدین صورت که اگر شماره ستون آن‌ها برابر باشد یا به صورت اریب همدیگر را بزنند.

تابع value : تعداد جفت وزیرهای که تابع threatening_states برای آن False برمی‌گرداند.

تابع neighbors : با فرض اینکه هربار همه وزیرها به جز یکی نمی‌توانند حرکت کنند ، و فقط آن وزیر حرکت می‌کند. برای مثال در حالت ۸×۸ هر استیت به تعداد ۷×۸=۵۶ همسایه دارد.

تابع hill_climbing : این تابع هربار همسایه های استیت را می‌گیرد و بین آن‌ها بهترین را انتخاب می‌کند ، اگر بهترین همسایه با استیت اولیه فیتنس برابری داشته باشد ، باز هم انتخاب می‌شود. این فرایند تا جایی ادامه می‌یابد که استیت بهتری بین همسایه‌ها نباشد.

تابع hill_climbing_random_restart : این تابع یک استیت اولیه

به صورت رندوم تولید می کند (به کمک تابع initial) و هر بار یک مرحله جلوتر رفته و بین همسایه‌ها بهترین را برمی‌دارد، و تست می کند که استیت نهایی است یا نه. یک limit هم دارد که می‌تونه به این تعداد مرحله جلو بره و اگر جواب را پیدا نکرد، منصرف شه.