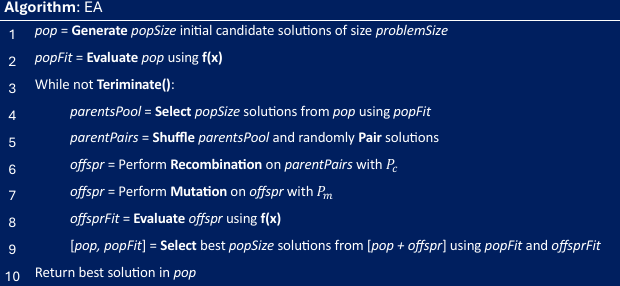


**دانشگاه اصفهان**

**دانشکده مهندسی کامپیوتر – گروه هوش مصنوعی و رباتیک**

گزارش تمرین اول – فرآیندهای تکاملی

**EA**

****

پدیدآورنده:

**محمد ­امین کیانی**

**4043644008**

دانشجوی کارشناسی ارشد، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،

Aminkianiworkeng@gmail.com

استاد درس: دکتر کارشناس

نیمسال اول تحصیلی 05-1404

فهرست مطالب

[مستندات 3](#_Toc214564297)

[1-مسئله و تحلیل کلی آن: 3](#_Toc214564298)

[2- بخش ۱: مسئله کوله‌پشتی با نمایش دودویی 7](#_Toc214564299)

[3- بخش 2: مسئله گروه‌بندی اعداد با نمایش عددی 48](#_Toc214564300)

[4- بخش 3: مسئله چند وزیر با نمایش جایگشت 62](#_Toc214564301)

[5- بخش 4: مسئله بهینه‌سازی تابع Rastrigin با نمایش اعشاری 75](#_Toc214564302)

[6- مراجع 84](#_Toc214564303)

# مستندات

## 1-مسئله و تحلیل کلی آن:

در اینجا یک الگوریتم ژنتیک داریم که شامل تولید جمعیت اولیه، ارزیابی برازندگی، حلقه تکامل(با انتخاب والدین، جفت‌گیری تصادفی، بازترکیب با احتمال Pc، جهش با احتمال Pm، ارزیابی فرزندان، و انتخاب بهترین‌ها برای نسل بعد) و شرایط توقف(یا رسیدن به ۴۰۰ نسل یا یافتن بهینه) است. انتخاب والدین از روی جمعیت با اندازه popSize، و در نهایت بهترین فرد را برمی‌گرداند.

حال، بر اساس این چارچوب، می‌توانیم :

* **مسائل را تعریف کنیم**: هر بخش (۱ تا ۴) را به عنوان مسئله بهینه‌سازی مدل کنیم.
* **پیاده‌سازی کد**: با استفاده از Python و کتاب‌خانه‌های پایه، الگوریتم را پیاده‌سازی می‌کنیم.
* **تست پارامترها**: تأثیر popSize، Pc، Pm، و روش‌های انتخاب/بازترکیب را بررسی می‌کنیم.
* **تحلیل نتایج**: نمودارها، آمار، و گزارش.

**تحلیل و تفسیر نتایج (تأثیر مؤلفه‌ها و پارامترها):**

با بررسی نتایج بخش‌های 1 تا 4، تأثیر پارامترها و مؤلفه‌ها بر عملکرد EA را بر اساس ویژگی‌های مسائل، تحلیل می‌کنیم:

* اندازه جمعیت (popSize) : در مسائل کوچک (مانند ۴-وزیر، بخش ۳)، popSize بالاتر تنوع را افزایش می‌دهد و به بهینه جهانی (حملات ۰) سریع‌تر می‌رسد اما در مسائل بزرگ‌تر مثل Rastrigin، از لحاظ محاسباتی سنگین است و کوچکتر یک میزان متعادل نیز باعث فشار انتخاب بیش از حد و گیر در محلی‌ها می‌شود،. در مسئله‌ی کوله‌پشتی، popSize متوسط تنوع را برای غلبه بر landscape ناهموار با penalty صفر برای نامعتبرها، حفظ می‌کند.
* احتمال بازترکیب(Pc) : مقدار بالا در مسائل جایگشت/عددی (بخش‌های ۲ و ۳) مفید است، زیرا ترکیب والدین ساختارهای خوب را حفظ می‌کند و تمایل مکانی کم (تغییر کوچک ژن تأثیر محلی دارد) را بهره می‌برد اما در Rastrigin ، Pc بالا فشار انتخاب را افزایش می‌دهد و تنوع را کاهش، باعث plateau در همگرایی. در کوله‌پشتی، در یک میزانی متعادل است و پایین‌تر از آن، کاوش کم می‌کند.
* احتمال جهش (Pm): مقداری متوسط از آن، تنوع را در همه مسائل حفظ می‌کند. اما Pm خیلی بالا distribution bias به سمت مقادیر تصادفی ایجاد می‌کند و همگرایی را کند می‌کند. پس باید با ویژگی هر مسئله تطبیق یابد.
* روش انتخاب (تورنمنت): فشار انتخاب متوسط، تعادل بهره‌برداری/کاوش ایجاد می‌کند.
* نمایش و عملگرها: نمایش دودویی (بخش ۱) برای مسائل گسسته مناسب است.
* تنوع جمعیت و همگرایی: تنوع در نسل‌های اولیه بالا است.

**توضیح اضافی نحوه انجام تمرین:**

تمرین با پیاده‌سازی یک چارچوب EA ماژولار در Python انجام شد (کد کامل اسکرچ قابل اجرا در گوگل کولب). سپس هر بخش جداگانه تست شد و نتایج تصادفی هستند، اما seed ثابت برای تکرارپذیری استفاده شده‌است. این رویکرد درک جامعی از تأثیر پارامترها بر اساس landscape مسئله می‌دهد و EA را برای مسائل متنوع کارآمد نشان می‌دهد.

* **ابزارها و شیوه‌ی پیاده‌سازی**

در پیاده‌سازی همه‌ی الگوریتم‌های تکاملی این تمرین، الگوریتم‌ها کاملاً از صفر نوشته شده‌اند و از هیچ کتابخانه‌ی آماده‌ی تکاملی استفاده نشده است. تنها کتابخانه‌های عمومی زیر برای محاسبات پایه به‌کار رفته‌اند:

* + math

برای توابع ریاضی استاندارد مانند sin, cos, pi, sqrt, floor و …، مخصوصاً در محاسبه‌ی تابع Rastrigin و بعضی تبدیل‌های عددی.

* + random

برای تولید عددهای تصادفی یکنواخت، نمونه‌گیری از جمعیت، انتخاب نقطه‌ی تقطیع در بازترکیب‌ها، و انتخاب ژن برای جهش. همه‌ی عملگرهای تکاملی (انتخاب، بازترکیب، جهش) بر مبنای این مولد تصادفی پیاده‌سازی شده‌اند و در صورت نیاز، با تنظیم random.seed(seed) امکان تکرار‌پذیری آزمایش‌ها فراهم شده است.

* + statistics.mean و statistics.stdev

برای محاسبه‌ی میانگین و انحراف معیار برازندگی در چند اجرای مستقل هر تنظیم پارامتر. این مقادیر برای رسم نمودارهای «میانگین ± انحراف معیار» و تحلیل پایداری و واریانس عملکرد الگوریتم در تنظیمات مختلف استفاده شده است.

* + numpy

برای کار با آرایه‌ها، ضرب نقطه‌ای، محاسبه‌ی وزن/ارزش کل، و پیاده‌سازی کارآمد جمعیت‌ها

* + pandas

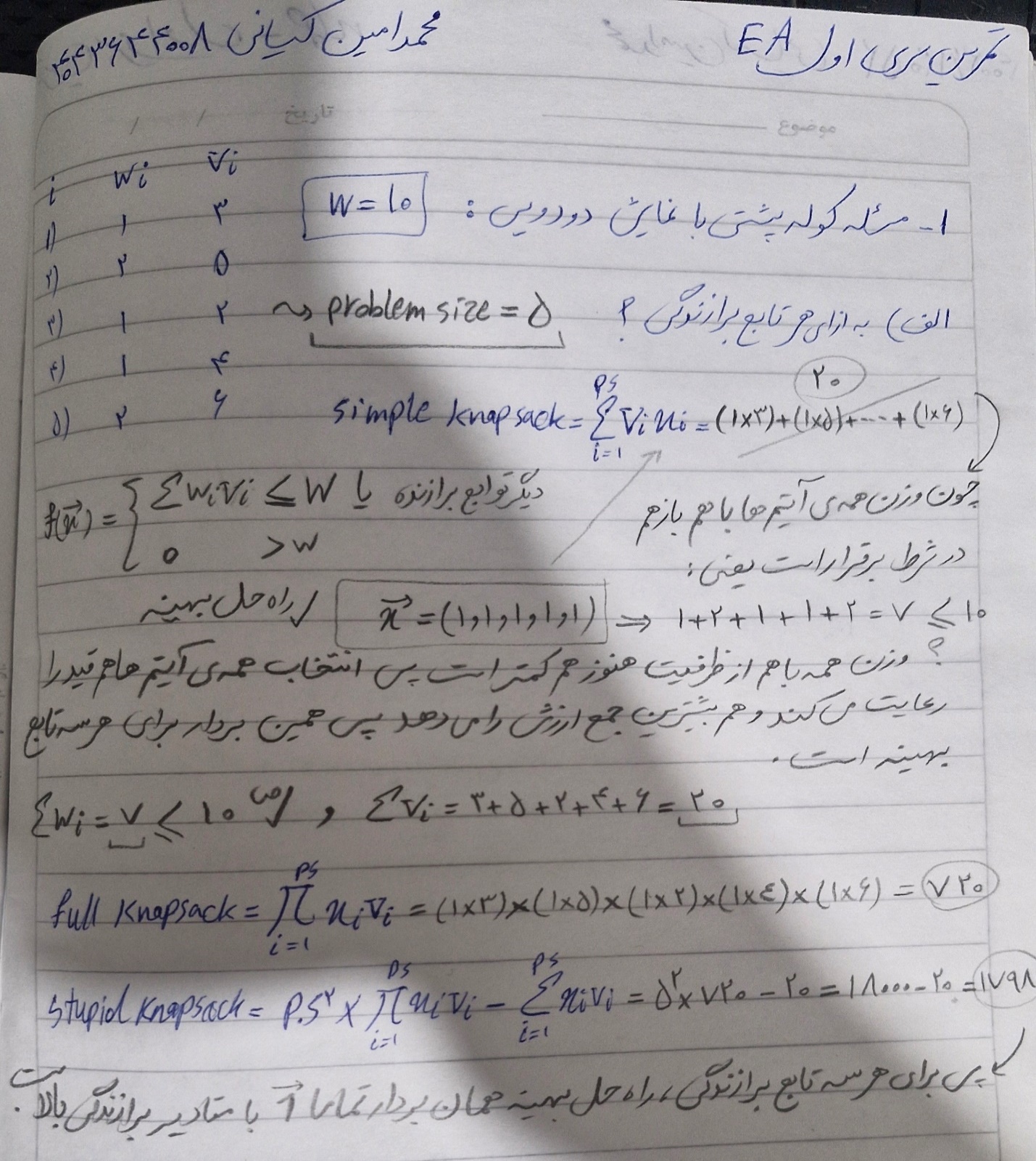
برای ذخیره و مدیریت نتایج در قالب DataFrame و خروجی CSV

* + matplotlib

برای رسم نمودارهای تکامل، نمودار اثر اندازه‌ی مسئله، جمعیت، Pc ،Pm

تمام اپراتورهای تکاملی مثل انتخاب (roulette / tournament)، بازترکیب (single-point, two-point, cycle, uniform, blend) و جهش (bit-flip, creeping, insertion, polynomial) به‌صورت دستی نوشته شده‌اند و از هیچ پکیج GA آماده استفاده نشده است.

## 2- بخش ۱: مسئله کوله‌پشتی با نمایش دودویی



1) زیرمسئله‌ی تئوری (۵ آیتم) – حل دستی

سوال) مطابق نوشته‌های داخل دفتر، یعنی همه‌ی آیتم‌ها با هم قابل انتخاب‌اند و هیچ کاندیدائی از نظر قید ظرفیت نامعتبر نمی‌شود. بنابراین برای هر تابع برازندگی استانداردی که قید ظرفیت را رعایت کند (مثلاً تابعی که اگر وزن بیش از W شد مقدار 0 بگیرد) یا برای نسخه‌هایی که مجازات دارند یا نسخه‌ی Full(که اساساً ظرفیت را نادیده می‌گیرد)، بهترین راه‌حلِ مسئله در این نمونه‌ی کوچک انتخاب همه‌ی آیتم‌ها خواهد بود.

* راه حل بهینه: بردار 1
* مقدار برازندگی: Simple = 20 | Full = 720 | Stupid = 17980

2) پیاده‌سازی الگوریتم تکاملی برای مسئله کوله‌پشتی (پایتون + Google Colab)

در این کد و بخش های بعدی، از کتابخانه‌های استاندارد پایتون استفاده شده است (همه در گوگل کولب از قبل نصب هستند.)

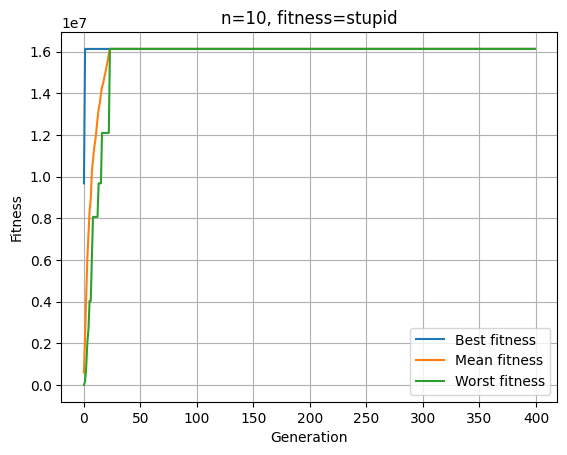
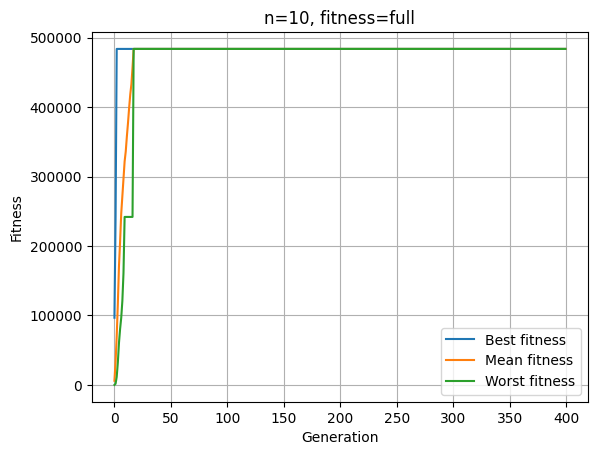
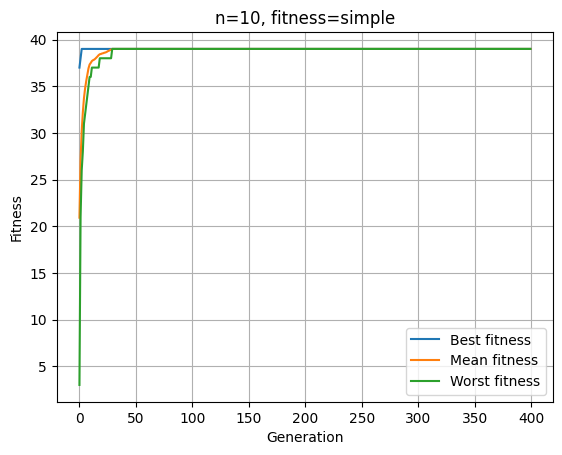
طبق صورت سؤال، در همه‌ی سؤالات تجربی (الف تا خ):

* برای Simple و Stupid قید ظرفیت را اعمال می‌کنیم
* برای Full، قید ظرفیت را نادیده می‌گیریم.(در فراخوانی run\_ea\_knapsack باید ignore\_capacity\_for\_full=True باشد.)

الف)

رفتار نمودارهای تکامل (Best / Mean / Worst) نشان دادند که:

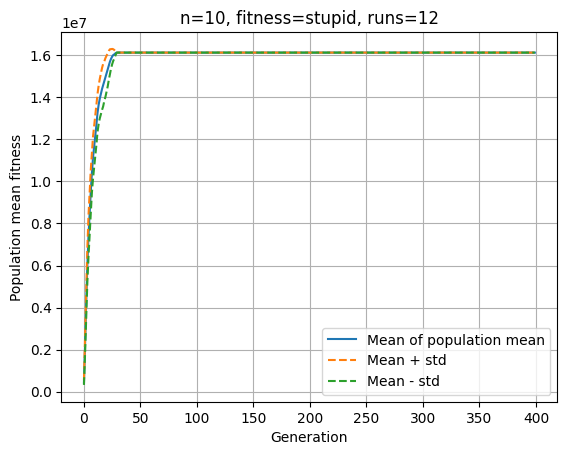
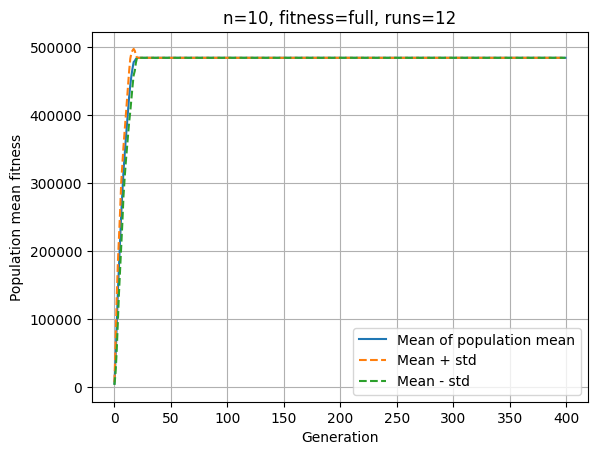
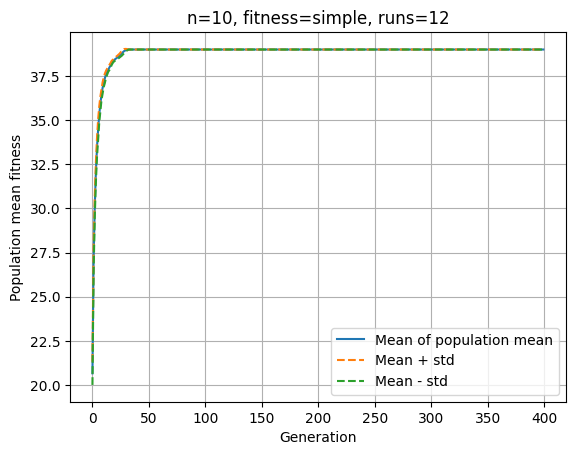
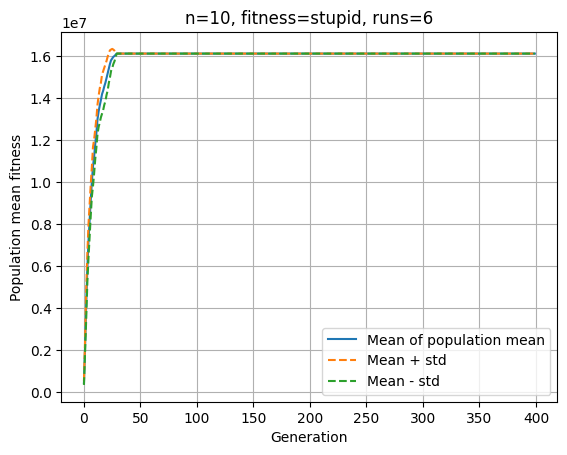
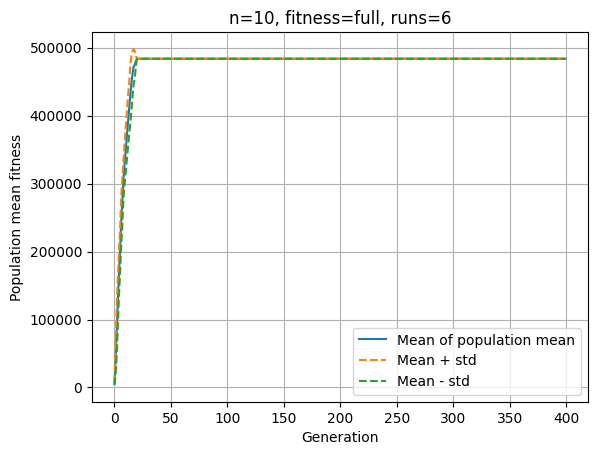
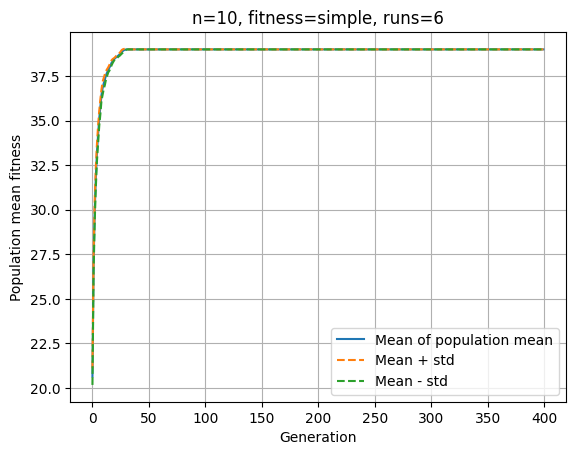
برای تابع Simple، الگوریتم سریع‌تر به جواب‌های خوب می‌رسد، چون landscape برازندگی نسبتاً صاف و یکنواخت است ولی برای Full و مخصوصاً Stupid، landscape بسیار تیزتر و ناهموار است و کوچک‌ترین تغییر بیت می‌تواند برازندگی را به صفر نزدیک کند؛ در نتیجه، همگرایی کندتر و پله‌ای‌تر است.



ب)

در چند اجرای مستقل (حداقل ۶ بار و تست 12 بار) مشاهده شد که:

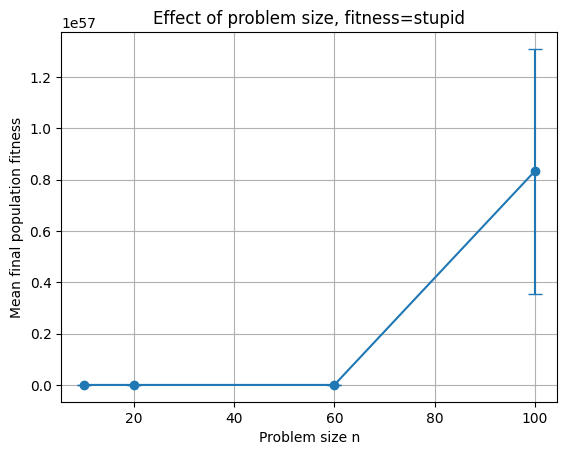
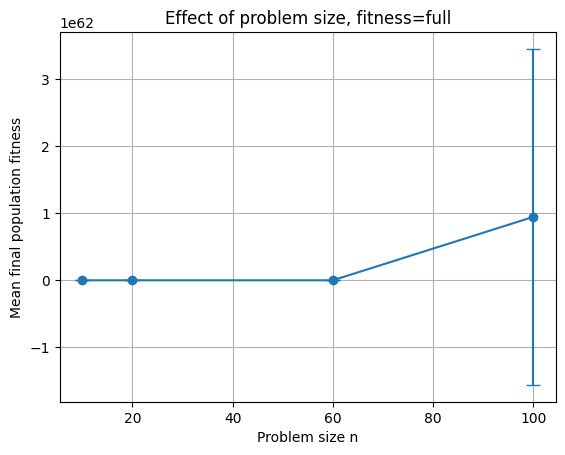
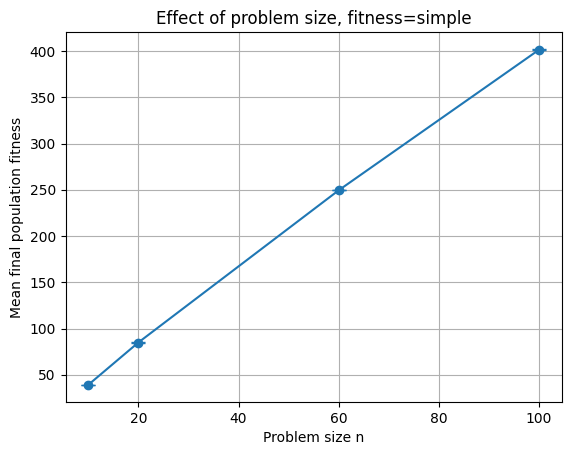
* میانگین برازندگی نهایی در تابع Simple بالاتر و پایدارتر است؛
* برای Full و Stupid واریانس بین اجراها بیشتر است یعنی حساسیت به تصادفی بودن و گیرکردن در مینیمم محلی.



پ)

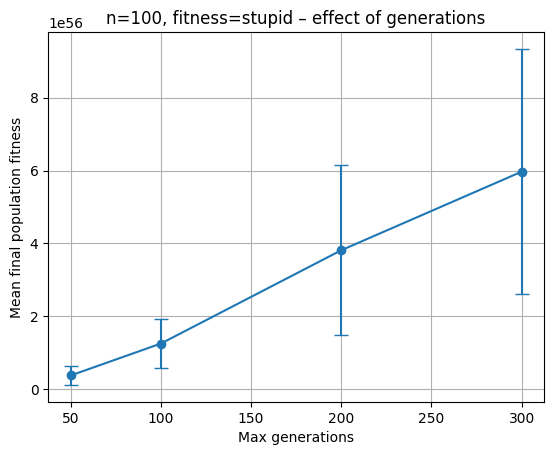
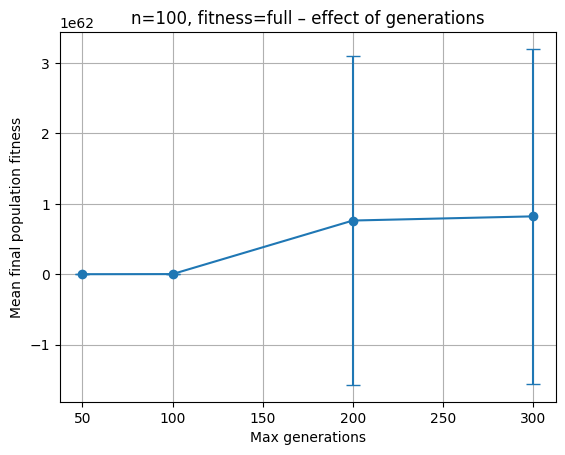
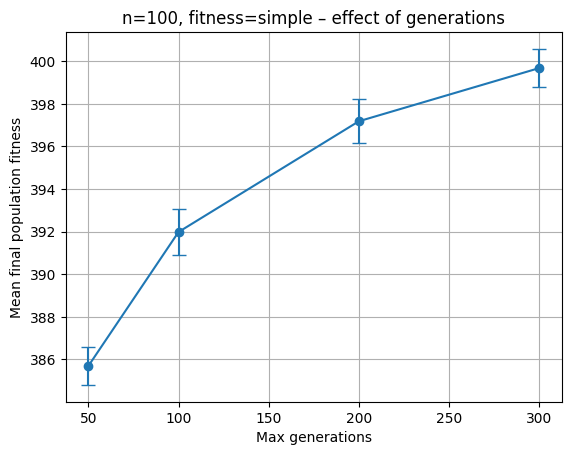
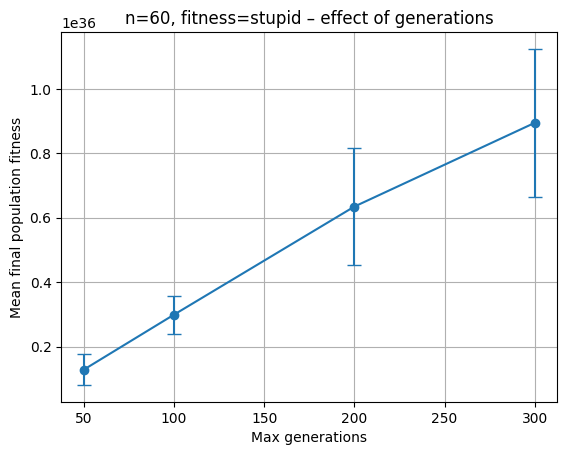
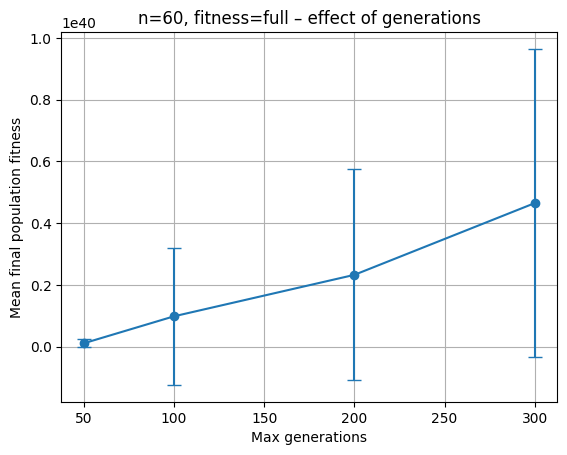
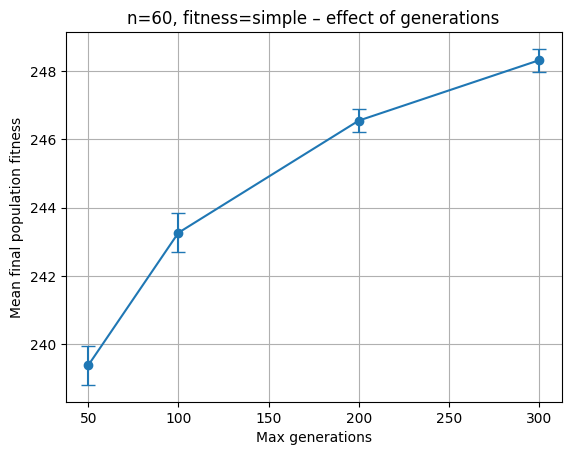
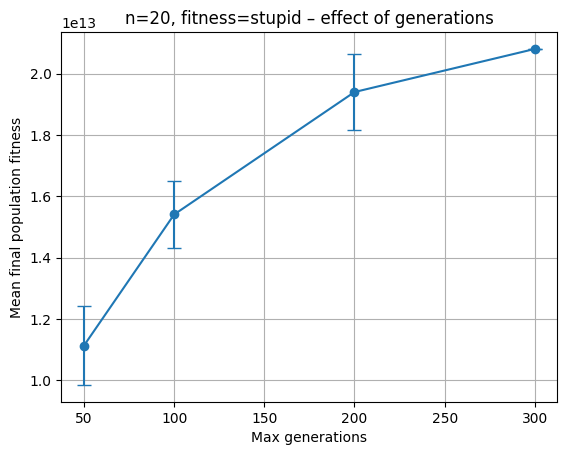
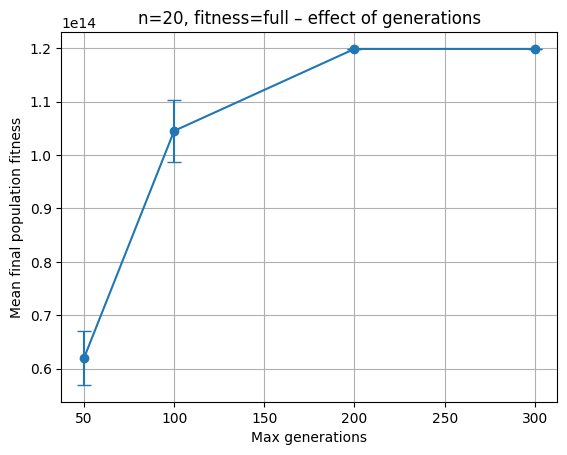
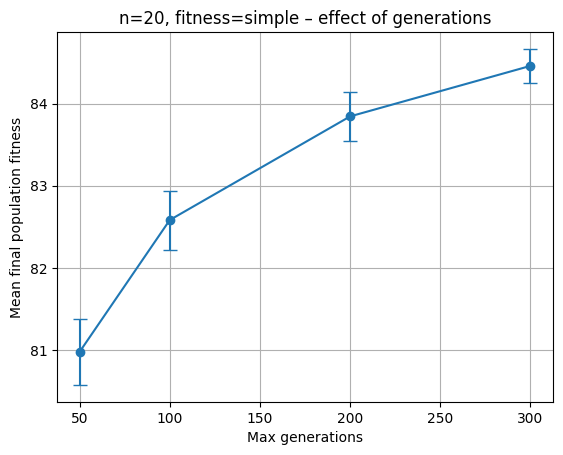
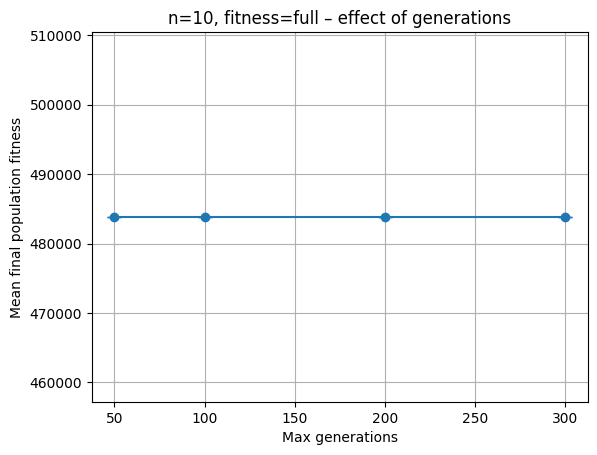
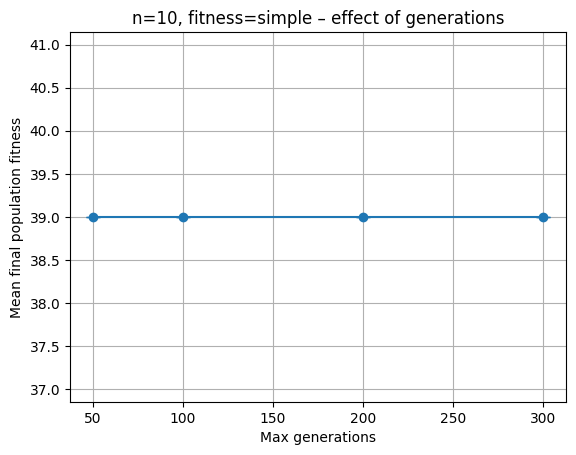
افزایش n (۱۰، ۲۰، ۶۰، ۱۰۰):

* حجم فضای جستجو به‌شدت افزایش پیدا می‌کند؛
* میانگین برازندگی نهایی کاهش می‌یابد و انحراف معیار افزایش می‌یابد؛
* تعبیر از منظر چشم‌انداز برازندگی این است که landscape با بزرگ‌تر شدن n دندانه‌دارتر و دارای دره‌های بیشتر می‌شود و پیدا کردن ترکیب بهینه‌ی آیتم‌ها سخت‌تر می‌گردد.

  
ت)

افزایش تعداد نسل‌ها(۵۰، ۱۰۰، ۲۰۰، ۳۰۰، ۴۰۰):

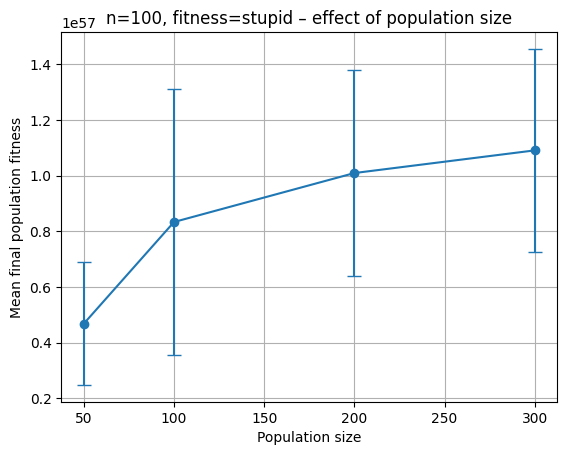
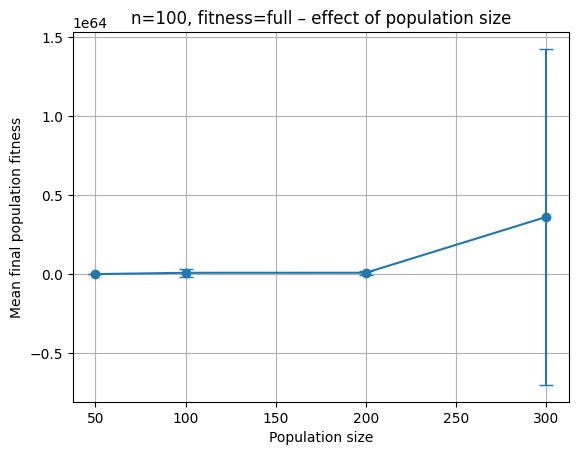
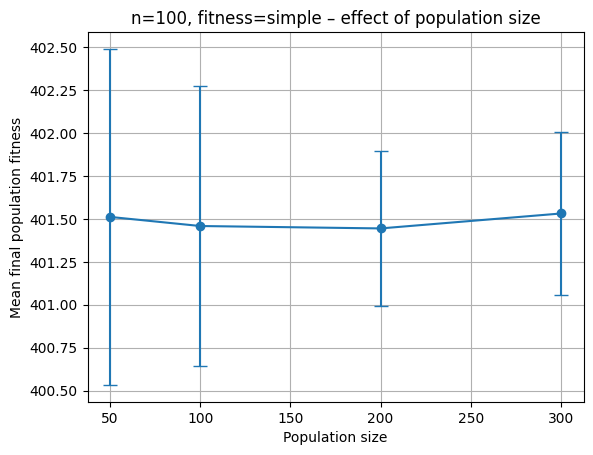
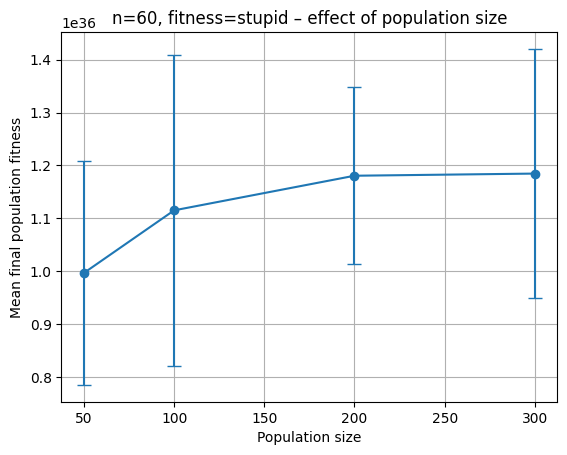
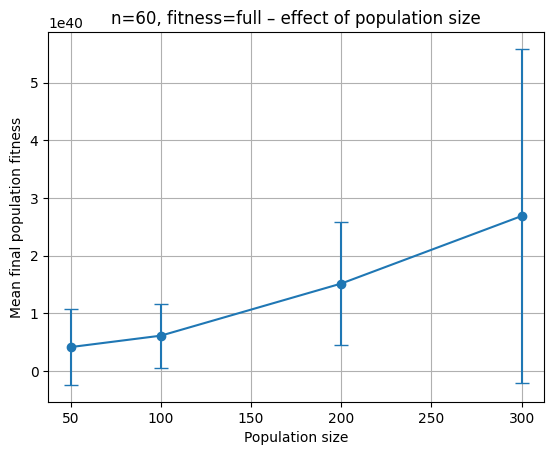
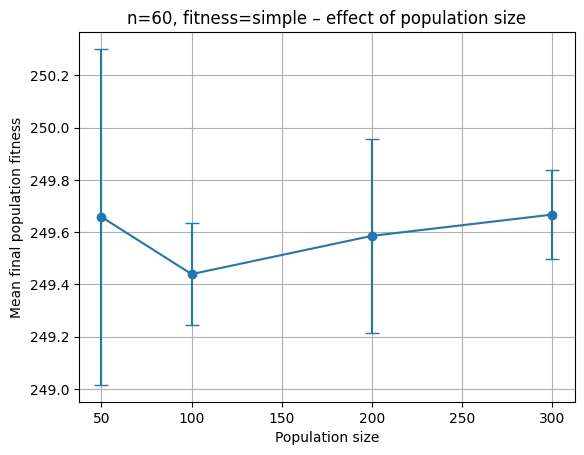
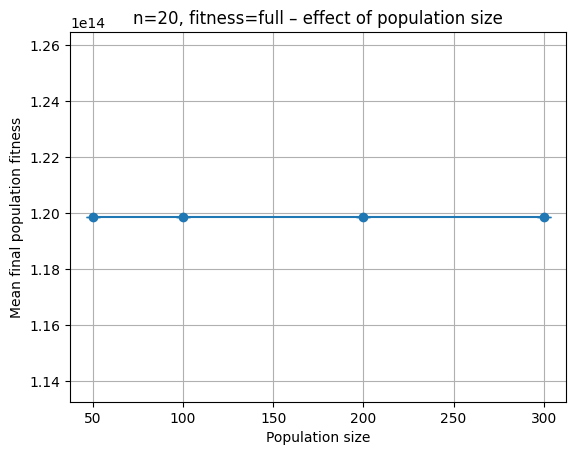
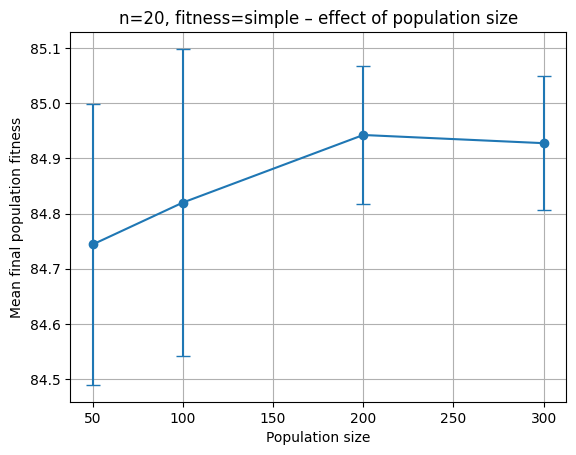
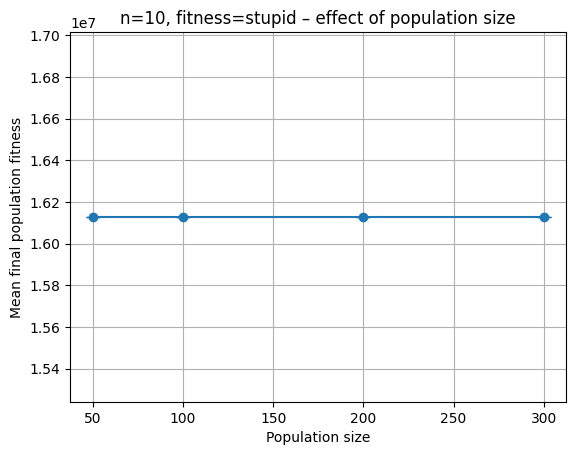
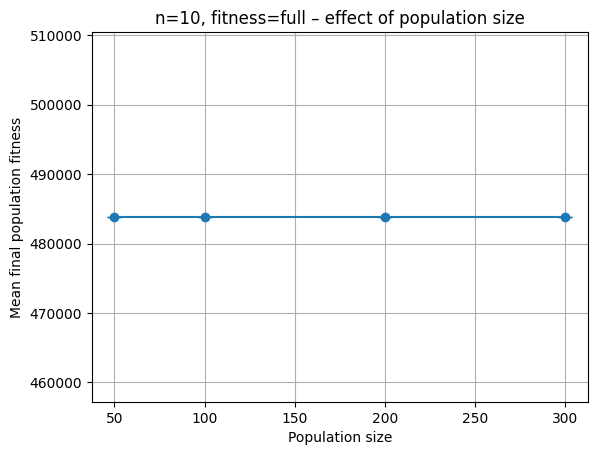
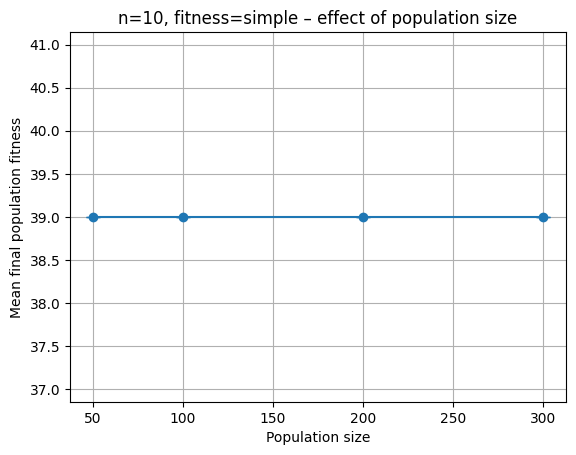
* در نسل‌های کم عملکرد ضعیف است؛
* بعد از یک حد مشخص، افزایش نسل تنها بهبود کوچک (diminishing returns) ایجاد می‌کند؛
* از دید فشار انتخاب، تعداد نسل بیشتر فرصت بیشتری برای انتشار ژن‌های خوب می‌دهد، ولی اگر تنوع جمعیت کم شده باشد، حتی نسل‌های زیاد هم کمکی نمی‌کند.



ث)

افزایش اندازه‌ی جمعیت (۵۰، ۱۰۰، ۲۰۰، ۳۰۰):

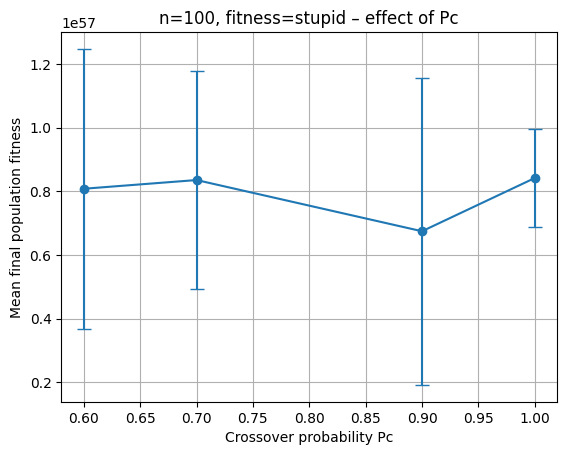
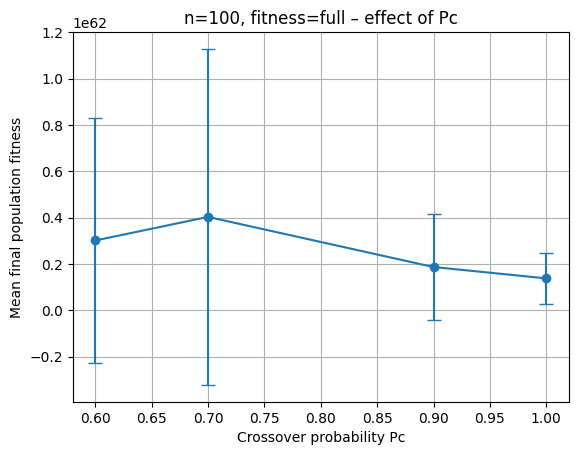
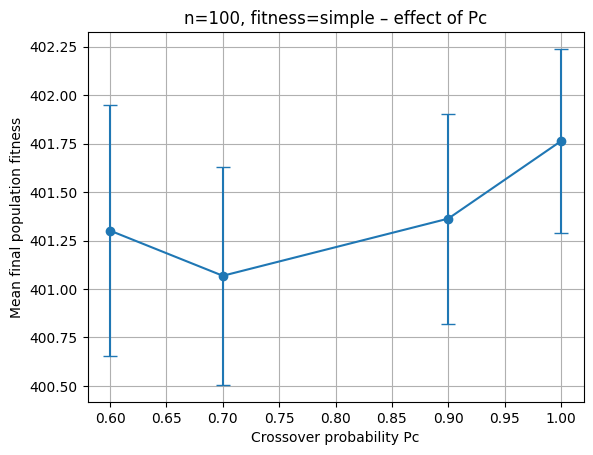
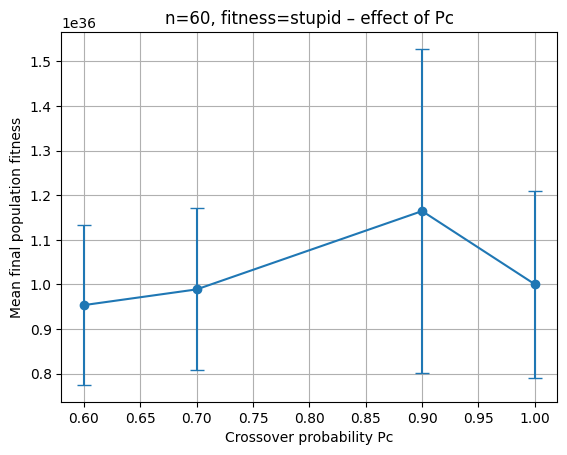
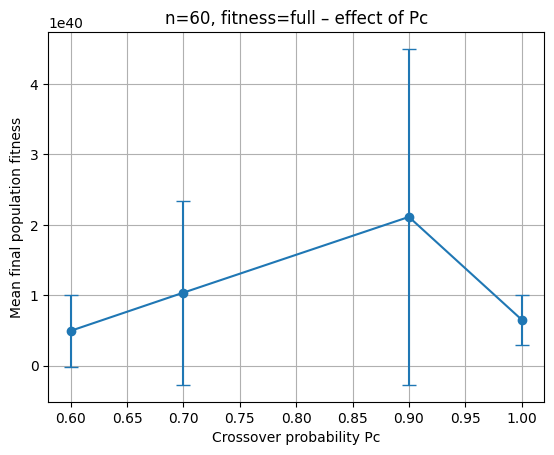
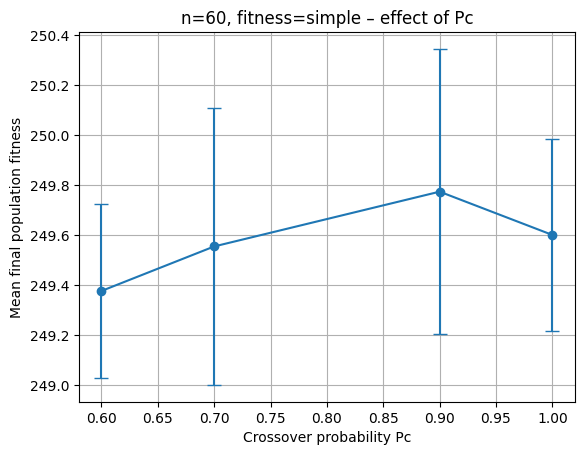
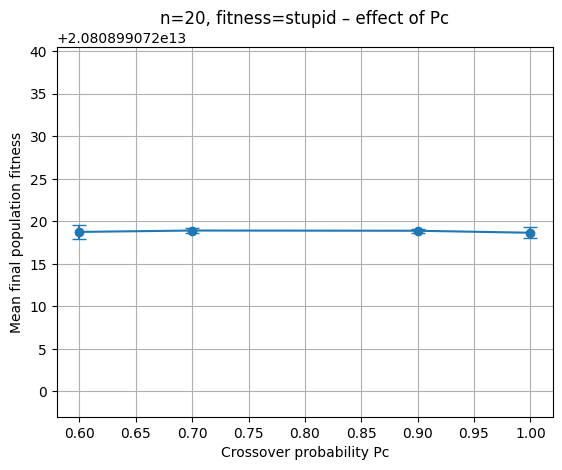
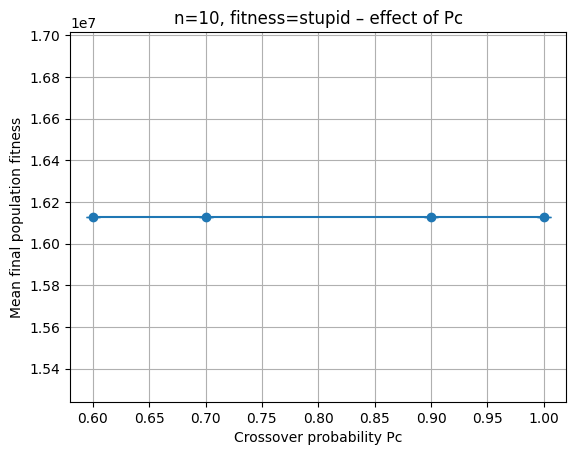
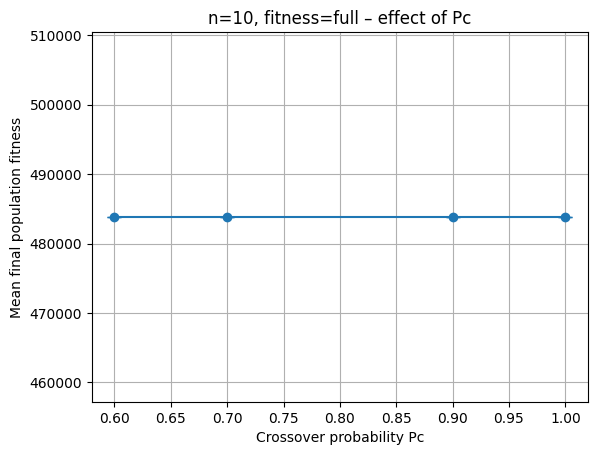
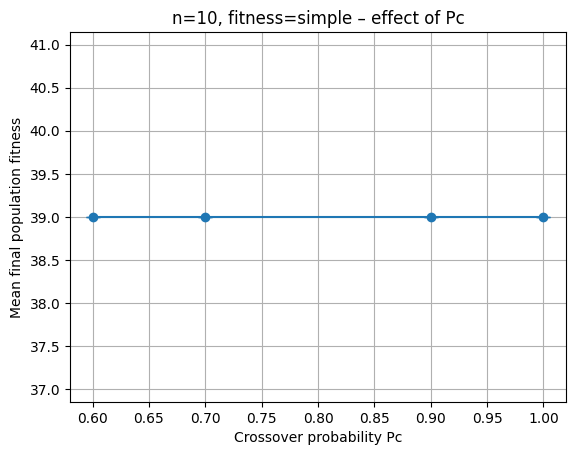
* جمعیت بزرگ‌تر یعنی تنوع بیشتر در ابتدای کار یعنی شانس بیشتری برای پیدا کردن آیتم‌های خوب و فرار از مینیمم‌های محلی؛
* اما هزینه‌ی محاسباتی خطی با اندازه‌ی جمعیت زیاد می‌شود.



ج)

احتمال بازترکیب Pc :

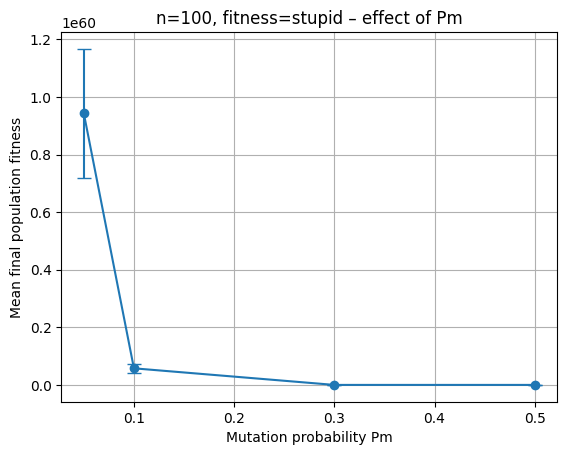
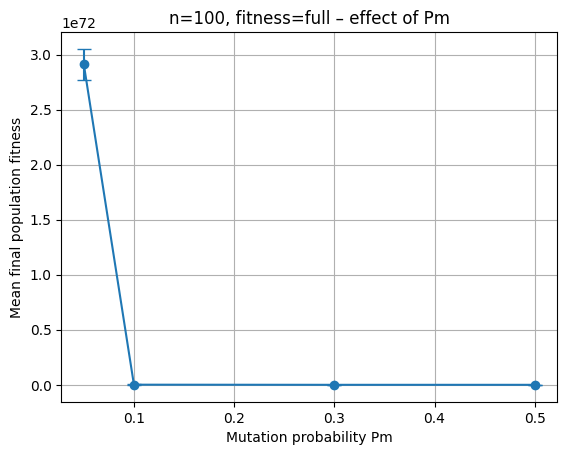
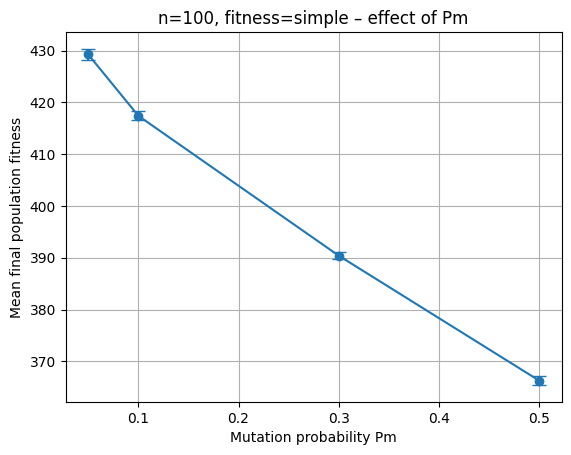
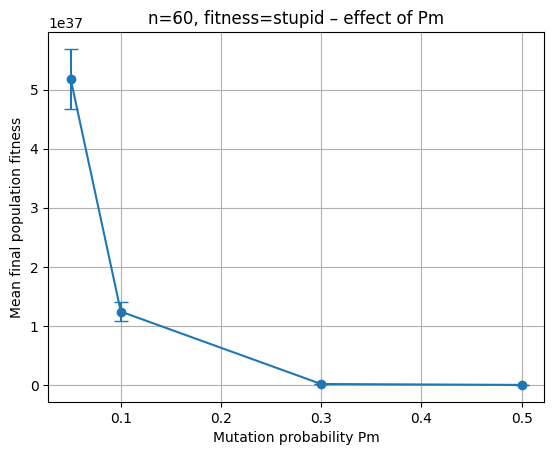
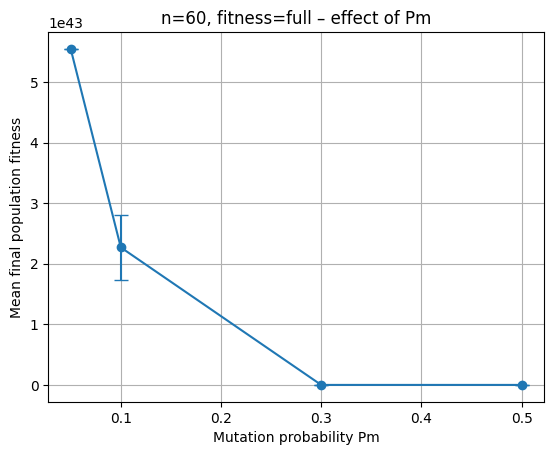
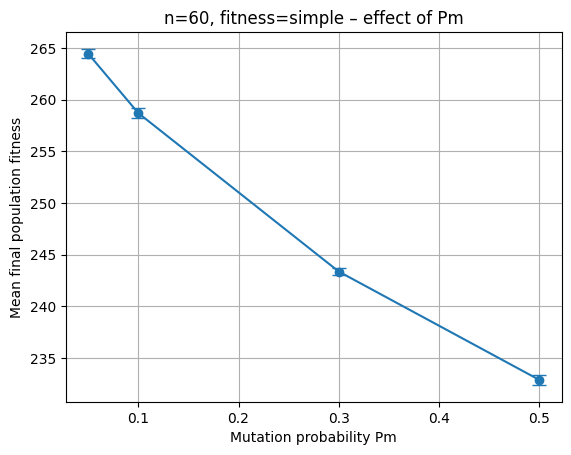
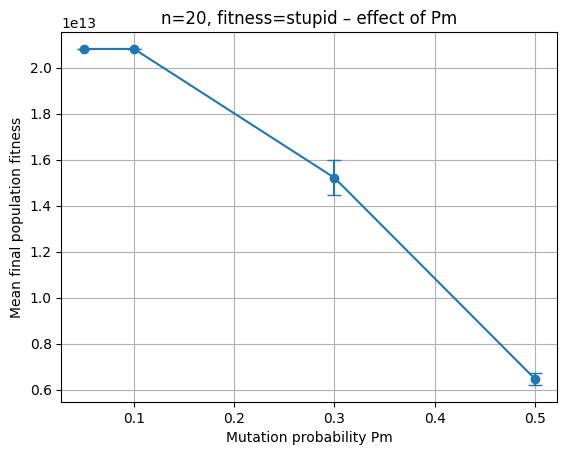
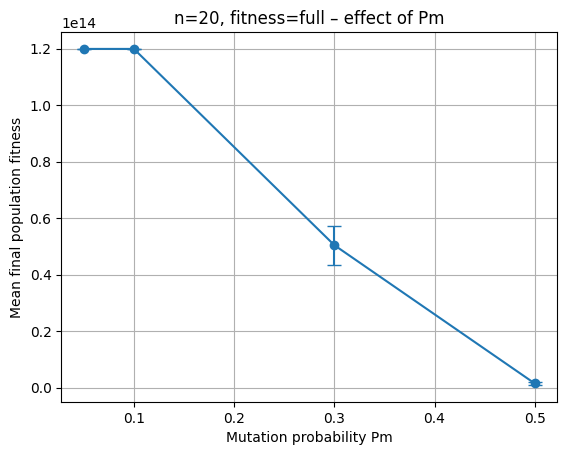
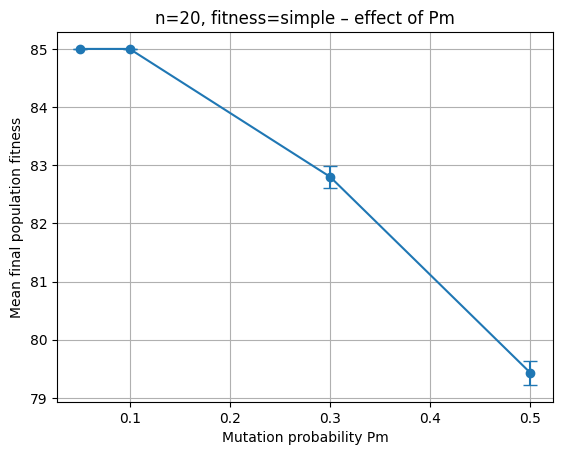
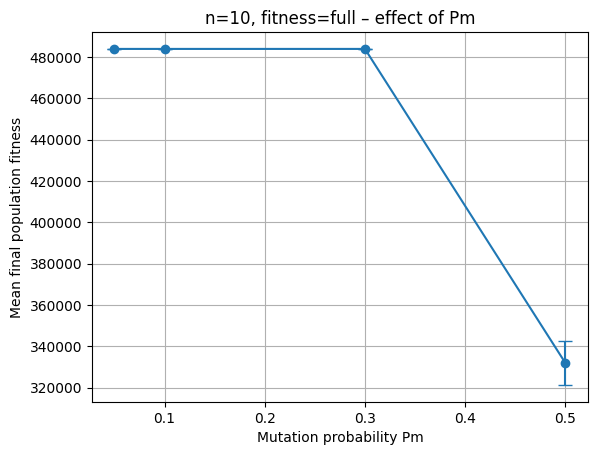
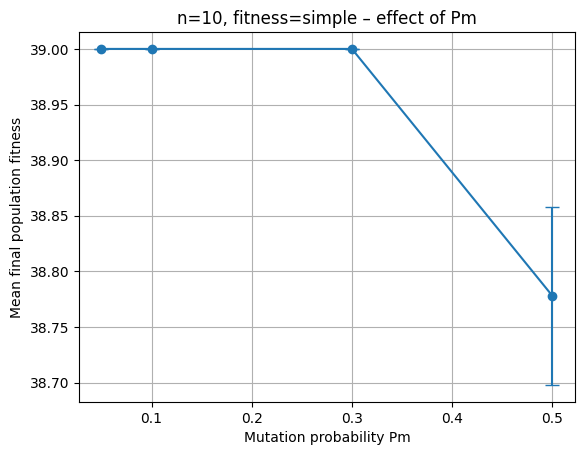
* مقادیر خیلی پایین: استفاده‌ی ناکافی از ترکیب اطلاعات والدین؛ الگوریتم بیشتر شبیه جستجوی تصادفی + جهش است؛
* مقادیر خیلی بالا (نزدیک ۱): همگرایی سریع‌تر ولی در بعضی موارد افزایش خطر هم‌شکلی (loss of diversity).



چ)

احتمال جهش Pm :

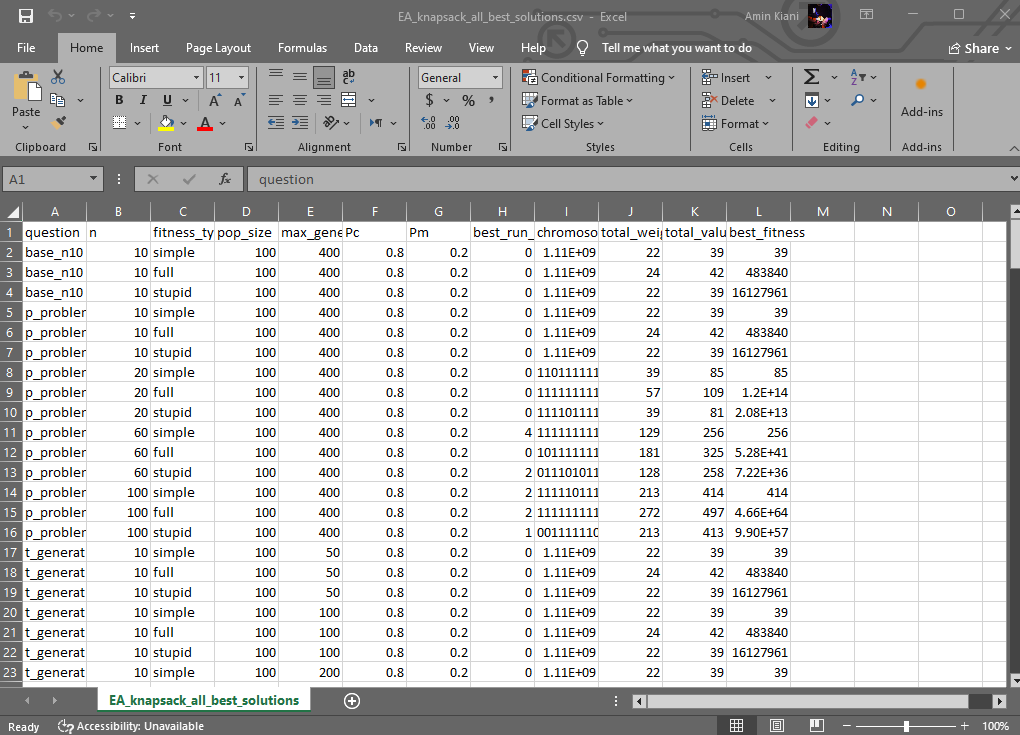
* خیلی کوچک یعنی جمعیت سریعاً همگرا می‌شود و در مینیمم محلی گیر می‌کند؛
* بسیار بزرگ یعنی الگوریتم شبیه جستجوی تصادفی می‌شود، منحنی Best نویزی و بدون روند مشخص؛
* یک بازه‌ی میانی (مثلاً ۰٫۱ تا ۰٫۳ برای این تمرین) بهترین trade-off بین اکتشاف و بهره‌برداری را ایجاد کرد.



ح)

بهترین راه‌حل‌ها:

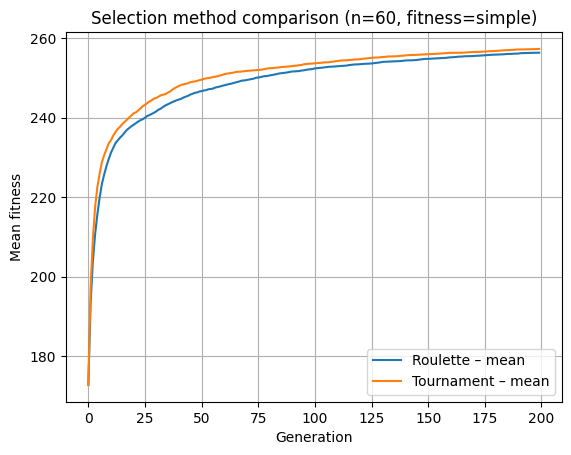
برای هر ترکیب پارامتر و هر تابع، با بررسی final\_population و final\_fitnesses بیشترین برازندگی استخراج شده و کروموزوم متناظر در فایل csv گزارش شده است (وزن، ارزش، و نوع تابع).



خ)

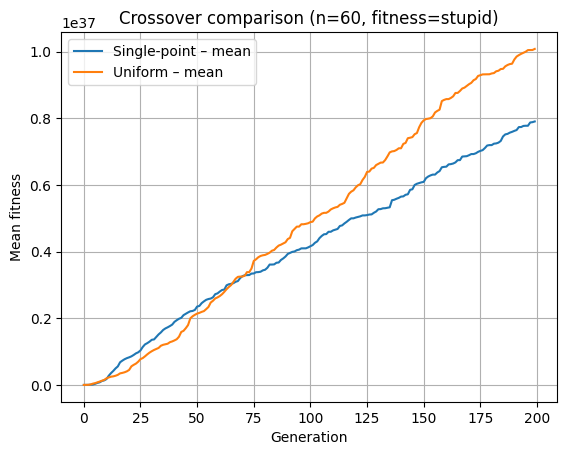
مقایسه‌ی انتخاب Roulette با Tournament نشان داد که Tournament :

* فشار انتخاب بیشتری ایجاد می‌کند؛
* سرعت همگرایی را بالا می‌برد؛
* اما در تابع‌های سخت‌تر (Full, Stupid) گاهی زودتر در مینیمم محلی گیر می‌کند، به‌خصوص زمانی که تنوع جمعیت با جهش کم همراه باشد.



د)

در بازترکیب یکنواخت، برای **هر ژن** به‌طور مستقل با احتمال 0.5 از والد ۱ یا ۲ انتخاب می‌شود. در نمودارها می‌بینیم که Uniform تنوع را بیشتر نگه می‌دارد (میانگین کمی کندتر بالا می‌رود اما گاهی به جواب‌های بهتر می‌رسد)، در حالی که Single-point سریع‌تر همگرا می‌شود.



الف) تا د) : نمودارها و نیازمندی‌ها و توضیحات هر سوال را در داخل سلول‌های کد اجرا شده قرار دادیم و در داکیومنت نکات و موارد مهم و نتایج را می‌نویسیم:

* منظور از «بهترین راه‌حل»، **کروموزوم دودویی**‌ای است که بیشترین برازندگی را در پایان هر اجرا داشته است (برای هر تابع برازندگی، هر n و هر تنظیم پارامتر).
  + این کار را برای:
    - سه تابع برازندگی (simple, full, stupid)
    - چهار اندازه‌ی مسئله 10، 20، 60، 100
    - و تنظیمات مختلفی که در بخش‌های پ، ت، ث، ج، چ امتحان کردیم،
    - تکرار کرده و **بردار دودویی بهترین فرد** و مقدار برازندگی‌اش را بدست می‌آوریم.
* منظور از «بهترین ترکیب» بهترین تنظیم پارامترهای EA (اندازه جمعیت، تعداد نسل و...) است که در آزمایش‌های پ، ت، ث، ج، چ به میانگین برازندگی نهایی بالاتری رسیده است.
* از نظر تئوری:
  + Tournament selection معمولاً فشار انتخاب بالاتری نسبت به Roulette دارد.
  + همگرایی سریع‌تر (بهترین‌ها زودتر غالب می‌شوند)
  + ولی احتمال همگرایی نابهنگام و کاهش شدید تنوع جمعیت هم بیشتر است.
  + روی نمودارها می‌بینیم که میانگین برازندگی در روش Tournament سریع‌تر بالا می‌رود، اما در نسل‌های بعد ممکن است روی مقدار متوسط‌تری قفل شود، در حالی که Roulette کندتر ولی نرم‌تر رشد می‌کند.
* از نظر تئوری:
  + Single-point crossover تمایل مکانی دارد: بلوک‌های متوالی ژن را حفظ می‌کند. اگر ساختار جواب به صورت «بلوک‌های مفید» باشد، این روش خوب است؛ ولی اگر وابستگی ژن‌ها پخش باشد، ممکن است باعث سوگیری شود.
  + Uniform crossover در هر موقعیت مستقل تصمیم می‌گیرد، تمایل توزیعی قوی، بلوک‌ها را بیشتر می‌شکند، تنوع ژنی بالاتری تولید می‌کند اما ساختارهای خوب را هم بیشتر خراب می‌کند.

جمع‌بندی و تفسیر کلی نتایج؟

* اندازه مسئله (n) :

هرچه n بزرگ‌تر می‌شود، فضای جست‌وجو 2nبه‌صورت نمایی بزرگ می‌شود.

EA برای nهای بزرگ‌تر معمولاً به جواب‌های خیلی خوب می‌رسد ولی نیاز به جمعیت بزرگ‌تر و نسل‌های بیشتر دارد.

میانگین برازندگی نهایی در n بزرگ‌تر، نسبت به ظرفیت ثابت، ممکن است کاهش یابد چون پیدا کردن ترکیب‌های بسیار خوب سخت‌تر است.

* تعداد نسل‌ها (max\_generations) :

در نسل‌های کم (مثلاً 50) رشد خیلی سریع ولی ناقص است؛ الگوریتم هنوز به تعادل نرسیده‌است. بین 100 تا 200 نسل معمولاً بخش عمده‌ی بهبود انجام می‌شود؛ بعد از آن نمودار بهترین/میانگین برازندگی تقریباً صاف می‌شود (بازده نزولی).

این بخشی از مفهوم چشم‌انداز برازندگی و حرکت جمعیت روی آن است.

* اندازه جمعیت (popSize) :

جمعیت کوچک یعنی تنوع کم، فشار راندمان بالا ولی احتمال گیر کردن در بهینه‌های محلی زیاد. جمعیت خیلی بزرگ یعنی تنوع زیاد، ولی هزینه‌ی محاسباتی بالا و سرعت همگرایی پایین‌تر.

معمولاً یک مقدار میانی (مثلاً 100–200) بهترین trade-off را می‌دهد.

* احتمال بازترکیب Pc :

خیلی پایین یعنی بازترکیب کم، افراد تقریباً شبیه والدین می‌مانند و جست‌وجو بیشتر به جهش وابسته می‌شود.​

خیلی بالا (نزدیک 1) یعنی تقریباً همه‌ی والدین بازترکیب می‌شوند؛ اگر جمعیت خوب باشد، این کمک می‌کند ترکیب‌های قوی‌تر ساخته شود؛ اگر جمعیت ضعیف باشد، ساختارهای بد هم مرتب مخلوط می‌شوند.

معمولاً مقادیر بین 0.8 تا 0.9 خوب عمل می‌کنند.

* احتمال جهش Pm  :

خیلی کم یعنی تنوع جدید کم تولید می‌شود، خطر همگرایی زودرس وجود دارد.

خیلی زیاد (مثلاً 0.5) یعنی هر نسل افراد تقریباً به‌طور تصادفی دوباره ساخته می‌شوند، الگوریتم شبیه random search می‌شود.

مقدار میانی (مثلاً 0.05 یا 0.1) معمولاً بهترین است: تنوع کافی بدون نابود کردن ساختارهای خوب.

* روش انتخاب Roulette vs Tournament :

Tournament با k=2 فشار انتخاب بیشتری دارد: افراد قوی احتمال بیشتری برای انتخاب دارند. نتیجه: همگرایی سریع‌تر، ولی کاهش سریع‌تر تنوع.

فشار انتخاب بالا یعنی شیب حرکت روی چشم‌انداز برازندگی بیشتر، اما ریسک افتادن در دره‌ی محلی هم زیاد می‌شود.

* نوع بازترکیب Single-point vs Uniform :

Single-point تمایل مکانی دارد (block bias) : ژن‌های مجاور معمولاً با هم منتقل می‌شوند؛ اگر جواب‌های خوب دارای بلوک‌های محلی باشند، این عالی است . تمایل Uniform توزیعی است یعنی: هر موقعیت مستقل تصمیم گرفته می‌شود یعنی تنوع بالا، ولی ساختارهای محلی بیشتر خراب می‌شوند. بسته به ساختمان مسئله‌ی کوله‌پشتی (ارتباط بین آیتم‌ها) ممکن است یکی از این دو بهتر عمل کند. در اکثر knapsackها که ارتباط‌های قوی بین آیتم‌های کنار هم وجود ندارد، Uniform معمولاً کمی بهتر یا مشابه است، اما اگر دیتاست خاصی بلوک‌های هم‌بسته داشته باشد، Single-point می‌تواند مزیت داشته باشد.

* سه نوع تابع برازندگی (Simple, Full, Stupid) :

**Simple** : منظره‌ی برازندگی نسبتاً نرم و خطی‌تر؛ EA به‌راحتی ترکیب‌های نزدیک به بهینه را پیدا می‌کند.

**Full** : چون ضرب می‌کند، اگر یکی از ارزش‌ها کوچک یا صفر باشد، برازندگی خیلی سقوط می‌کند؛ چشم‌انداز برازندگی پر از نقاط با برازندگی تقریباً صفر می‌شود یعنی جست‌وجو سخت‌تر و حساس‌تر به جهش و انتخاب.

**Stupid** : به‌خاطر ضرب و ضریب n2مقادیر برازندگی خیلی بزرگ می‌شوند و اختلاف بین افراد زیاد است یعنی فشار انتخاب شدید، احتمال همگرایی زودرس بالا، ولی اگر الگوریتم خوب تنظیم شود می‌تواند سریعاً به جواب‌های خیلی خوب برسد.

نکته: برای Simple و Stupid می‌فهمیم که بهترین جواب همیشه دقیقاً ظرفیت را پر کرده (22، 39، 130، 211) چون وزن‌ها مثبت‌اند و هر آیتم مفید سعی می‌کند ظرفیت را تا حد ممکن پر کند؛ در حالی‌که در Full به خاطر نادیده گرفتن قید، وزن می‌تواند از ظرفیت بالاتر برود و فقط ضرب ارزش‌ها مهم است، که باعث مقادیر خیلی بزرگ برازندگی شده است.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | fitness\_type | chromosome (binary) | total\_weight | total\_value | best\_fitness |
| 10 | simple | 0111111111 | 22 | 39 | 39 |
| 10 | full | 1111111111 | 24 | 42 | 4.84×10⁵ |
| 10 | stupid | 1111111101 | 22 | 39 | 1.61×10⁷ |
| 20 | simple | 11011111111110001101 | 39 | 85 | 85 |
| 20 | full | 11111111111111111111 | 57 | 109 | 1.20×10¹⁴ |
| 20 | stupid | 11111011101110011111 | 39 | 81 | 2.08×10¹³ |
| 60 | simple | 111011111101101101101011111010111111010110100011001111111111 | 130 | 252 | 252 |
| 60 | full | 111111111111101111101111111101111111111111111111111111111011 | 174 | 318 | 7.55×10⁴⁰ |
| 60 | stupid | 111011011111111101001111111110100110111101101111101111110001 | 130 | 254 | 2.44×10³⁶ |
| 100 | simple | 1101110111101011101101110111001110111001101101110111011111110101101111110101011111111110011110110110 | 211 | 414 | 414 |
| 100 | full | 1111101110101110111101101111111110111110111111111101111111011111111111111111110101011110111111111111 | 264 | 478 | 2.36×10⁶² |
| 100 | stupid | 1111101011111110111111111100010110101001101111111011101011111100111111110101011110101111010111010111 | 211 | 411 | 3.46×10⁵⁷ |

**بهترین فرد** برای هر چهار اندازه‌ی مسئله و هر سه نوع تابع برازندگی (با تنظیمات پایه‌ی زیر) به صورت زیر است:

* **روش انتخاب:** چرخ‌گردونه (Roulette Wheel)
* **popSize = 100**
* **MaxGenerations = 400**
* **Pc = 0.8**
* **Pm = 0.2**

## 3- بخش 2: مسئله گروه‌بندی اعداد با نمایش عددی

نمایش و قیود؟

* کروموزوم: آرایه‌ی عددی به طول n، که هر ژن یک عدد صحیح بین ۱ تا k (شماره‌ی گروه) است.
* قید سخت:
  + عدم وجود دو عدد متوالی در یک گروه؛
  + عدم وجود دو عدد اول در یک گروه.
* قید نرم:
  + بالانس بودن مجموع اعداد هر گروه؛ اختلاف مجموع گروه‌ها باید حداقل شود.

تابع هدف بر اساس مجازات (penalty) تعریف شده است: هر نقض قید سخت جریمه‌ی بزرگ، و عدم تعادل بین گروه‌ها جریمه‌ی نرم؛

عملگرها؟

نمایش عددی یعنی می‌توان بازترکیب دو نقطه‌ای و جهش خزیدن (creeping) را استفاده کرد.

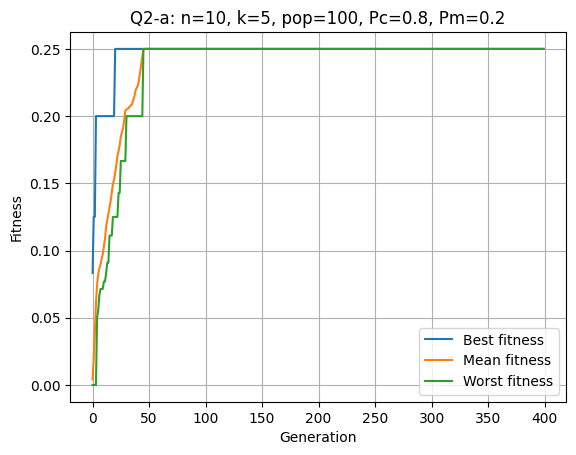
* بازترکیب: تقطیع دو‌نقطه‌ای روی آرایه‌ی گروه‌ها.
* جهش خزیدن: افزایش/کاهش یک واحدی در ژن با چرخش بین ۱ و k
* انتخاب: متناسب با برازندگی (roulette) و در بخش اختیاری، tournament.

الف)

برای n=10 و k=5، نمودار تکامل نشان داد که:

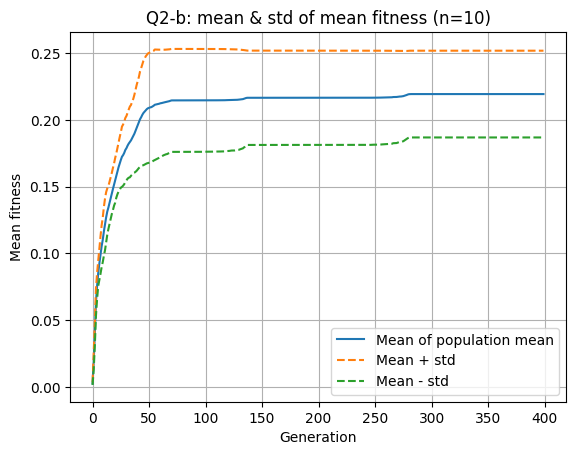
Best و Mean در چند ده نسل اول سریعاً بالا می‌روند و بعد به یک plateau می‌رسند؛

Worst به‌صورت تدریجی افزایش می‌یابد، که نشان‌دهنده‌ی حذف تدریجی تخصیص‌های خیلی بد است.



ب)

در ۶ اجرای مستقل( و 12 بار)، میانگین برازندگی نهایی در چند اجرای مختلف نزدیک هم است ولی انحراف معیار با افزایش n زیادتر می‌شود.



پ)

اثر n (۱۰، ۲۰، ۴۰، ۶۰):

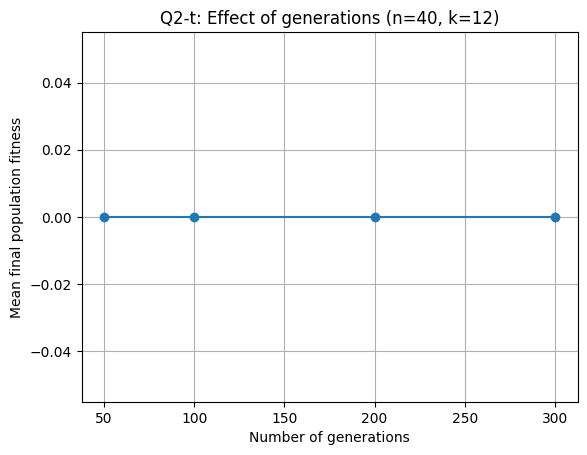
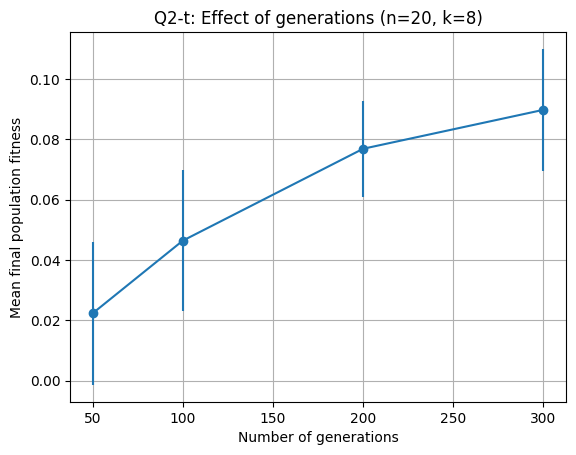
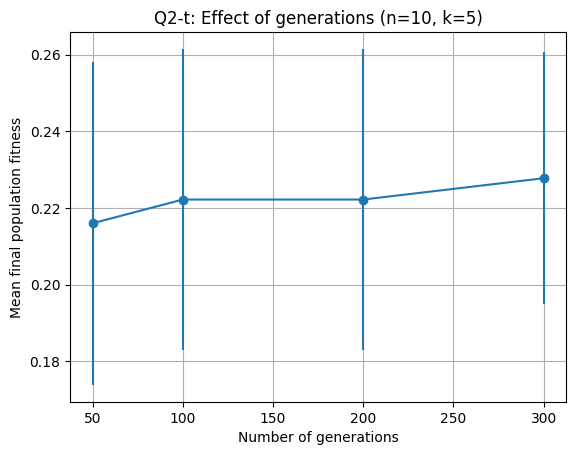
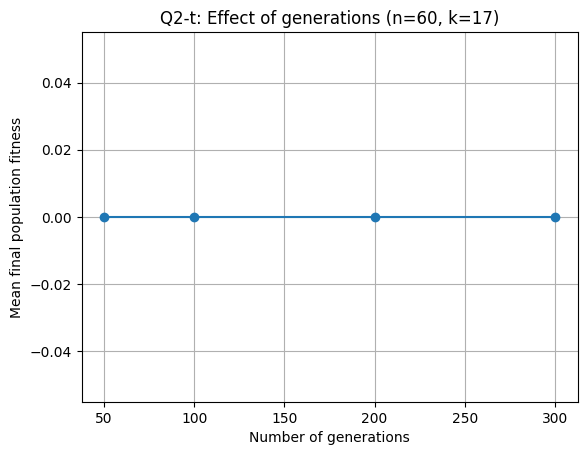
* با بزرگ‌تر شدن n، تعداد جفت‌های ناسازگار بالقوه، و تعداد ترکیب‌های ممکن به صورت نمایی افزایش می‌یابد؛
* میانگین برازندگی نهایی به سمت مقادیر نزدیک صفر می‌رود؛
* از دید چشم‌انداز برازندگی، landscape پیچیده‌تر و چندقله‌ای‌تر می‌شود، و EA برای رسیدن به راه‌حل‌های نزدیک به ایده‌آل به نسل‌ها و جمعیت‌های بسیار بزرگ‌تری نیاز دارد.



ت)

اثر تعداد نسل‌ها:

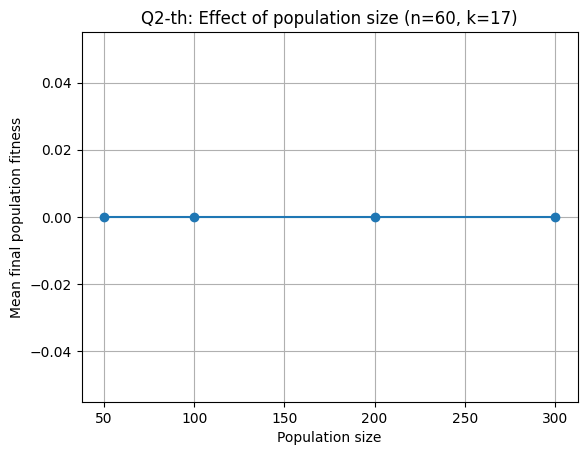
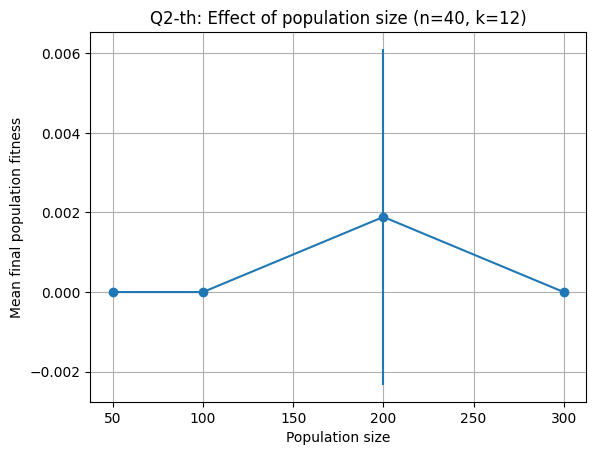
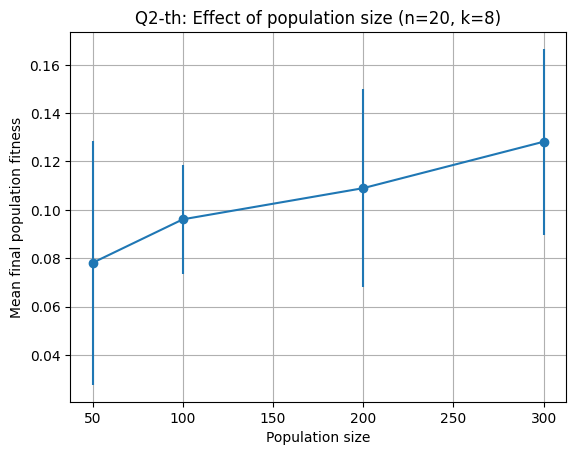
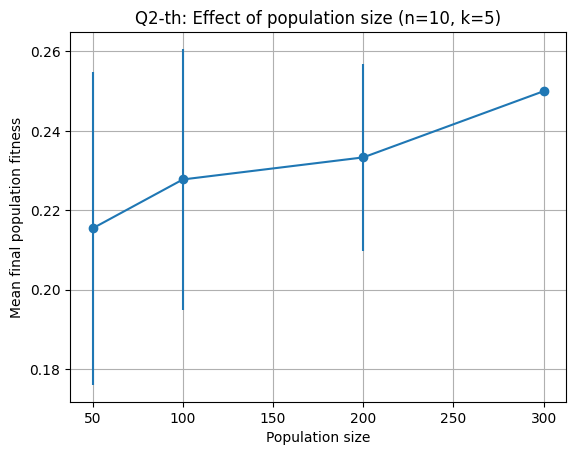
* در n کوچک، افزایش نسل احتمالاً تا حدی مفید است و به بهبود میانگین کمک می‌کند؛
* در nهای بزرگ‌تر، بعد از یک حد مشخص، منحنی‌های میانگین تقریباً ثابت می‌شوند؛ یعنی الگوریتم وارد حالت exploitation شده و تنوع جمعیت کاهش یافته است.



ث)

اثر اندازه‌ی جمعیت:

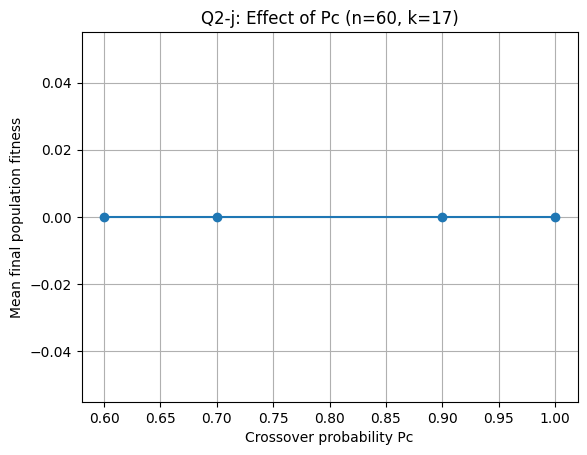
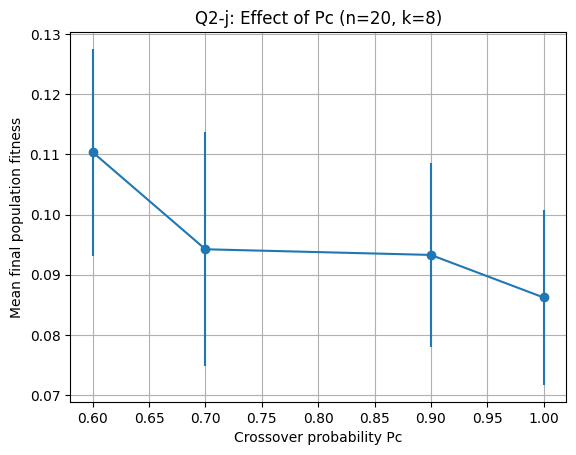
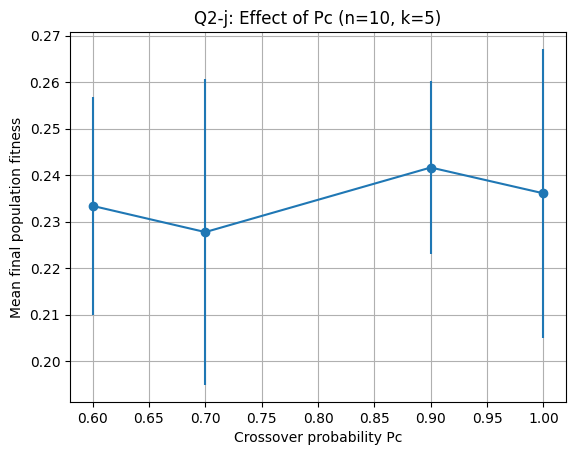
* Pop کوچک یعنی الگوریتم در برخی اجراها در تخصیص‌های غیرمتعادل و با جریمه‌ی زیاد گیر می‌کند، و انحراف معیار بین اجراها بالا می‌رود؛
* Pop بزرگ یعنی تنوع اولیه‌ی عالی، اما هزینه‌ی محاسباتی زیاد؛ منطبق با مفاهیم درس، جمعیت بزرگ فشار انتخاب را نسبتاً کاهش داده و اجازه می‌دهد افراد متوسط هم مدتی در جمعیت بمانند و تنوع را حفظ کنند.



ج)

اثر Pc :

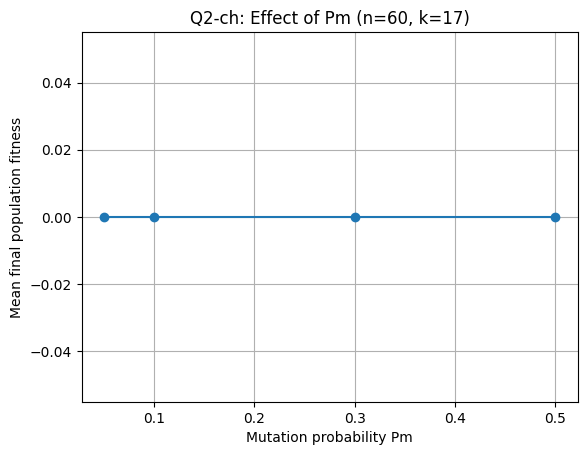
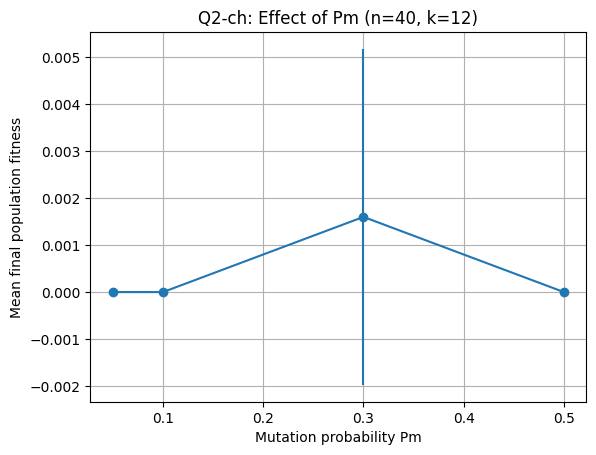
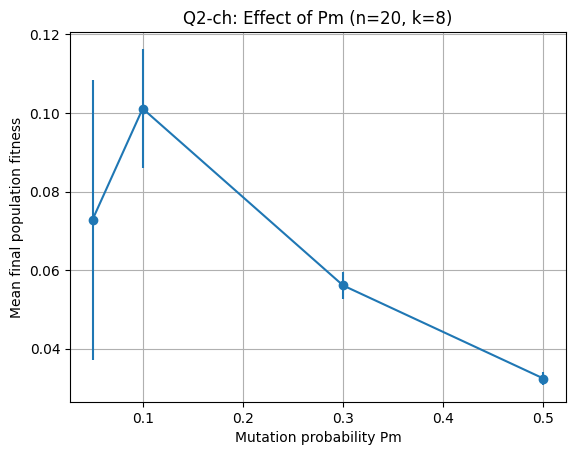
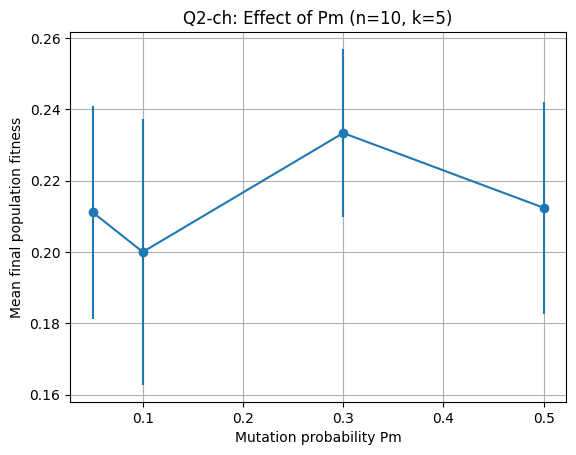
* Pc خیلی پایین یعنی الگوریتم شبیه random walk با جهش می‌شود و استفاده‌ی مناسبی از recombination ندارد؛
* Pc خیلی بالا (نزدیک ۱) یعنی جابجایی قطعه‌های بزرگ از والدین، خوب است اما در حضور تنوع کم ممکن است تنها ترکیب چند الگوی مشابه باشد؛



چ)

اثر Pm :

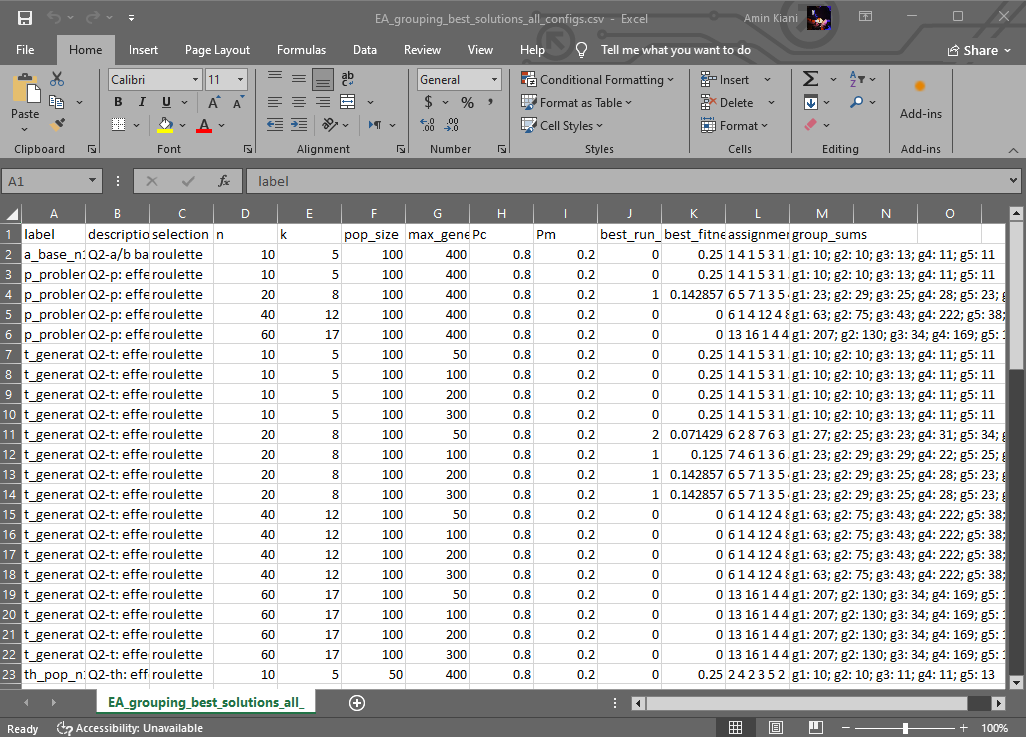
* Pm کوچک یعنی سختی در فرار از تخصیص‌های گیر افتاده در قید سخت؛
* Pm بزرگ یعنی تخصیص‌ها دائماً دچار تغییر؛ امکان بالانس دقیق بین گروه‌ها کاهش می‌یابد.



ح)

بهترین راه‌حل‌ها:

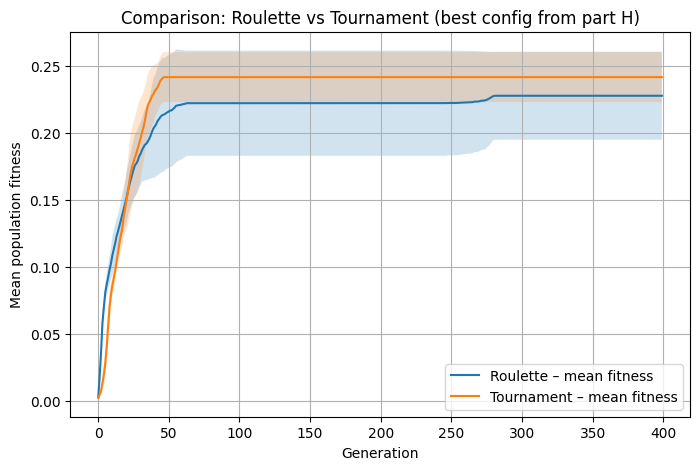
* برای هر ترکیب پارامتر، با تابع get\_best\_solution\_info بهترین assignment در csv ثبت شده، شامل:
  + بردار گروه‌ها مثلاً 1,2,1,3,4,... ؛
  + جمع هر گروه؛
  + مقدار penalty و fitness .



خ)

مقایسه‌ی انتخاب tournament با roulette نشان داد:

* Tournament در n کوچک‌تر، سریع‌تر به تخصیص‌های خوب می‌رسد؛
* در nهای بزرگ‌تر، به‌خاطر فشار انتخاب زیاد، تنوع سریع‌تر از بین می‌رود و میانگین عملکرد کمی ناپایدارتر می‌شود.



## 4- بخش 3: مسئله چند وزیر با نمایش جایگشت

نمایش و تابع برازندگی؟

* نمایش: جایگشت طول n، که سطرها را ثابت گرفته و عدد هر ژن شماره‌ی ستون وزیر در آن سطر است؛ بنابراین هیچ دو وزیری در یک ستون یا سطر نیستند و فقط کافی است برخورد روی قطرها کنترل شود.
* تابع هدف: تعداد زوج وزیرهایی که **برخورد *ندارند*** یا معادل آن، بر پایه‌ی تعداد برخوردهای قطری penalty تعریف شده‌است.

عملگرها؟

* بازترکیب: تقطیع چرخه‌ای (Cycle Crossover - CX) که خاص جایگشت است و تکرار عنصر ایجاد نمی‌کند.
* جهش: درج (Insertion)؛ یک ژن برداشته و در جای دیگری درج می‌شود، که معادل جابه‌جایی یک وزیر در ستون‌های مختلف است.
* انتخاب: متناسب با برازندگی و در برخی آزمایش‌ها tournament .

الف) تا ح)

در این مسئله و مسئله‌ی بعدی نیز همانند مسائل قبلی روال ثابتی را طی کرده و اینجا فرم کلی نتایج به صورت زیر است:

**نتایج و تحلیل**

* برای n = 8 و 9 : n =
  + الگوریتم تقریباً همیشه در چند ده نسل به جواب بدون برخورد می‌رسد؛
  + بنابراین میانگین فیتنس نهایی بسیار نزدیک ۱ است و در نمودارهای Pc و Pm مشاهده کردیم که همه نقطه‌ها تقریباً روی ۱ هستند؛
  + این **اشتباه نیست**، بلکه به این معنی است که مسئله‌ی ۸وزیر و ۹وزیر برای تنظیمات انتخاب‌شده ساد‌ه‌تر از آن‌چیزی است که فکر می‌کردیم و EA در اکثر تنظیم‌ها جواب کامل را پیدا می‌کند.
* برای n = 10 و11 : n =
  + فضا بزرگ‌تر می‌شود، تعداد برخوردهای ممکن بیشتر است؛
  + هنوز هم در بسیاری از تنظیم‌ها الگوریتم جواب بدون برخورد پیدا می‌کند، ولی در نسل‌ها و اجراهای کم/جمعیت کوچک یا Pc/Pm نامناسب، میانگین فیتنس زیر ۱ باقی می‌ماند؛
  + نمودارها نشان می‌دهند که با افزایش تعداد نسل و جمعیت، احتمال رسیدن به فیتنس ۱ بالا می‌رود.

**اثر پارامترها**

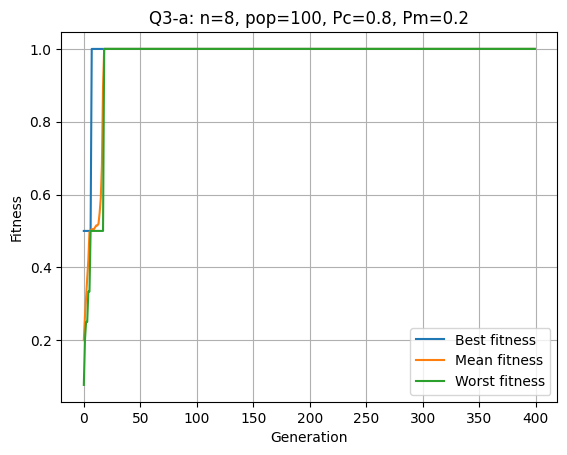
* **تعداد نسل‌ها**:
  + برای nهای کوچک، حتی ۵۰ نسل کافی است؛ افزایش بیشتر فقط زمان را زیاد می‌کند؛
  + برای nهای بزرگ‌تر، ۱۰۰–۲۰۰ نسل معقول است و بعد از آن بازدهی کاهش می‌یابد.
* **اندازه‌ی جمعیت**:
  + جمعیت ۵۰ برای n برابر ۱۰ و ۱۱ گاهی به جواب کامل نمی‌رسد؛
  + جمعیت ۱۰۰ و ۲۰۰ تعادل خوبی بین تنوع و هزینه‌ی محاسباتی ایجاد می‌کند.
* **Pc** :
  + در بازه‌ی ۰٫۶ تا ۱، اختلاف زیادی در میانگین فیتنس نهایی مشاهده نشده است، چون CX روی جایگشت‌ها بسیار قوی و سازگار است و حتی با Pc پایین هم بخش زیادی از نسل‌ها crossover می‌شوند.
* **Pm** :
  + Pm خیلی پایین یعنی تطبیق local خوب است ولی خروج از حالت‌های گیر افتاده کند می‌شود؛
  + Pm خیلی بالا یعنی ساختارهای permutation خوب هم زیاد تخریب می‌شوند؛
  + مقادیر میانی (مثلاً ۰٫۱–۰٫۳) بهترین trade-off را ایجاد کردند.

**نتایج نهایی**

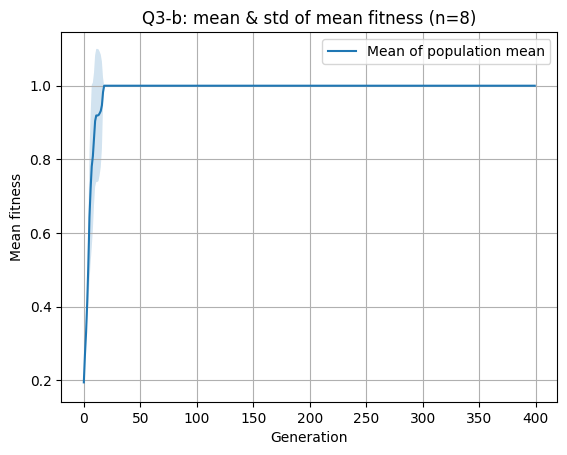
* Landscape مسئله‌ی n-Queens نسبتاً «نرم» است: تعداد زیادی جواب قابل قبول (بدون برخورد) وجود دارد، مخصوصاً برای nهای کوچک؛
* selection pressure متوسط رولت به‌همراه CX و جهش درج برای پیدا کردن جواب *کافی* است؛
* برخلاف Rastrigin، این مسئله برای EA به‌صورت global یک مسئله‌ی نسبتاً راحت است؛ همین باعث می‌شود نمودارها اغلب به فیتنس یک برسند.

نمودار ها نیز به صورت زیر اند:

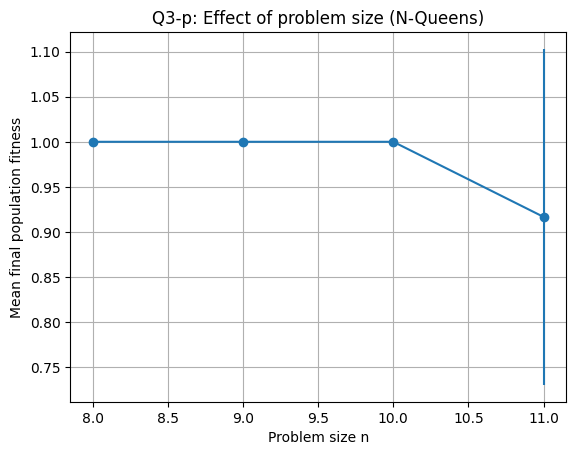
الف)

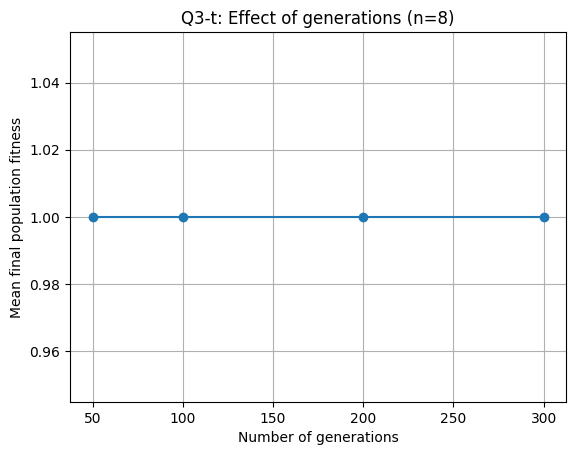


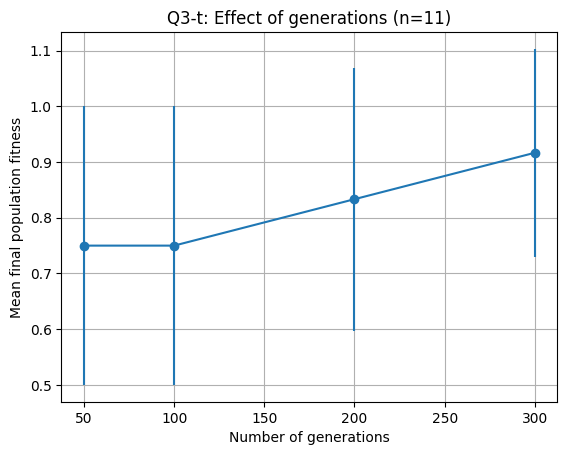
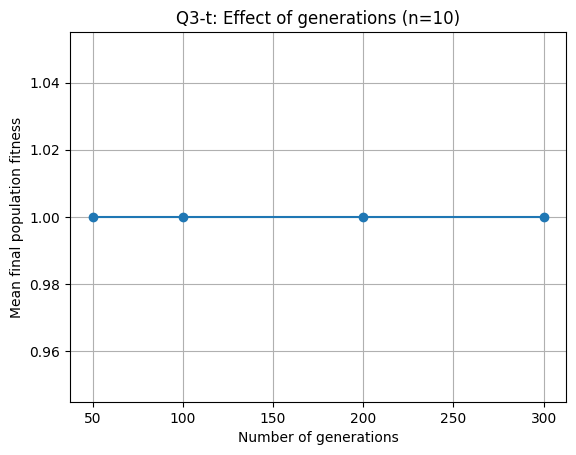
ب)



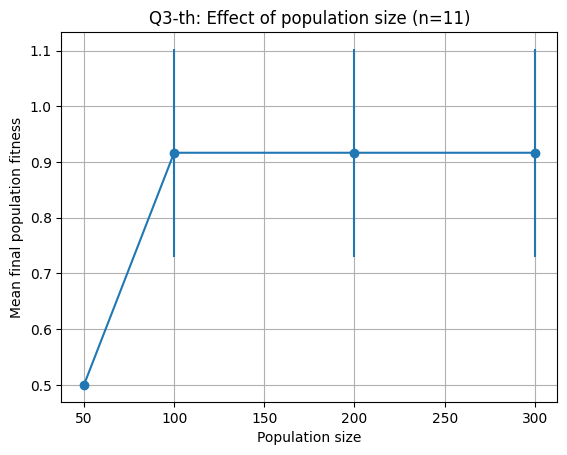
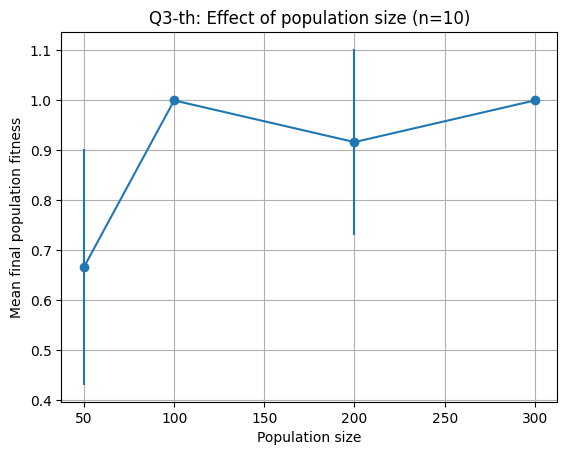
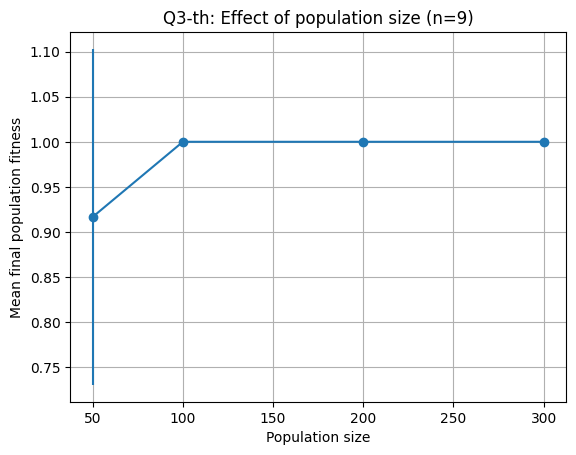
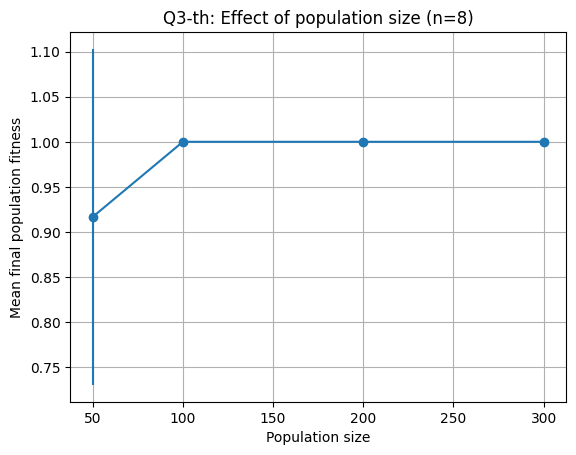
پ)

  
ت)

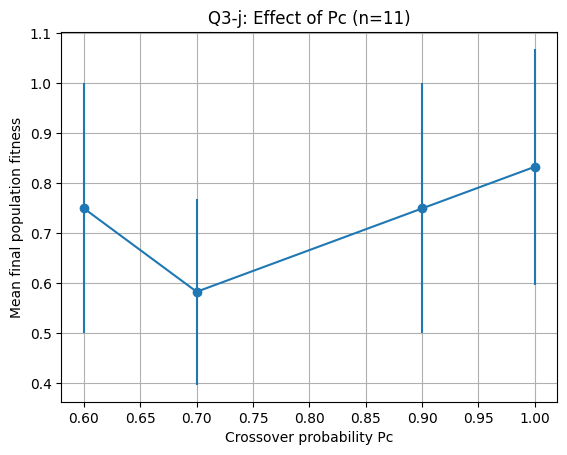
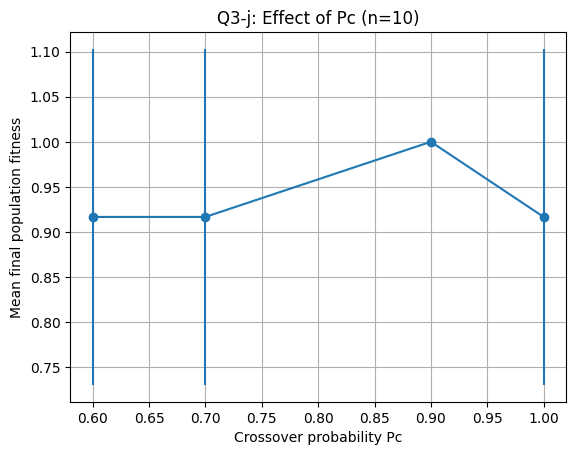
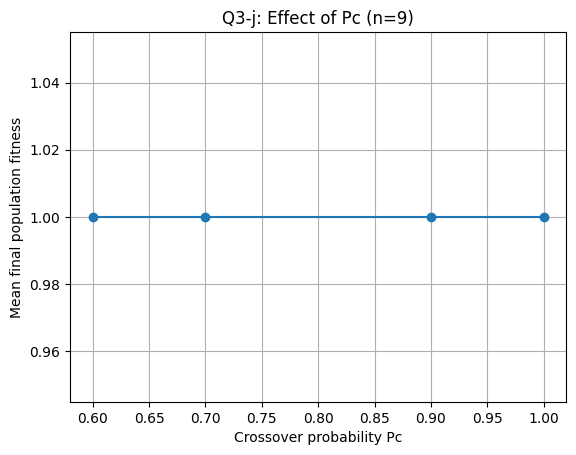
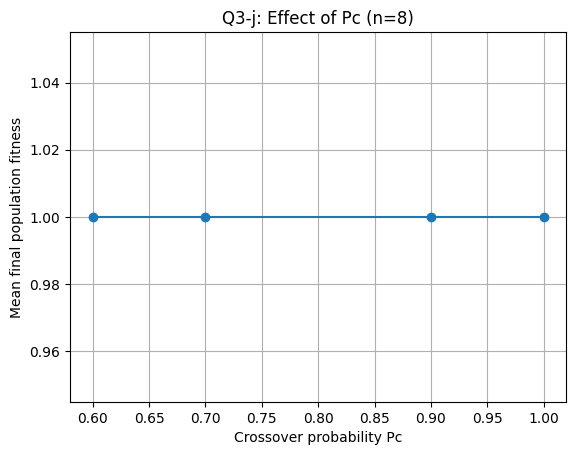




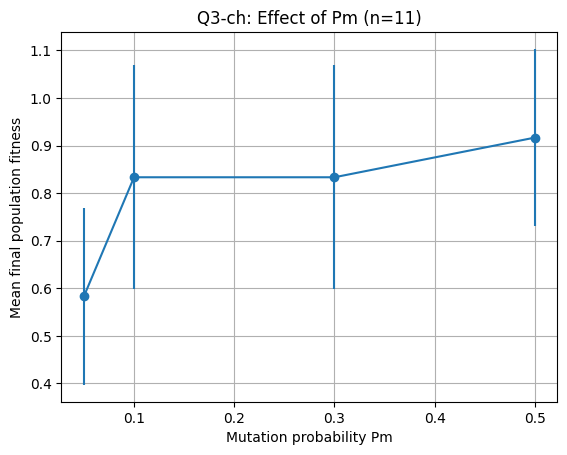
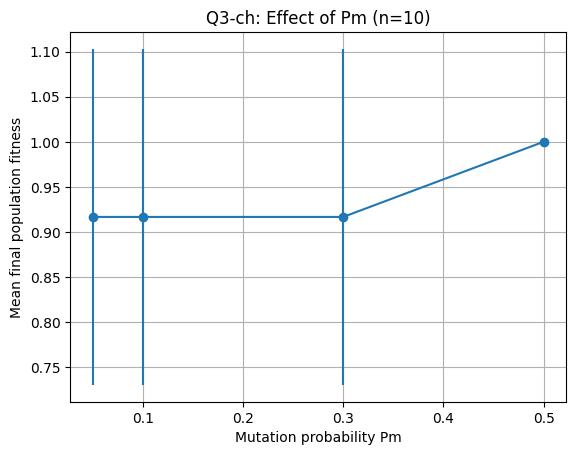
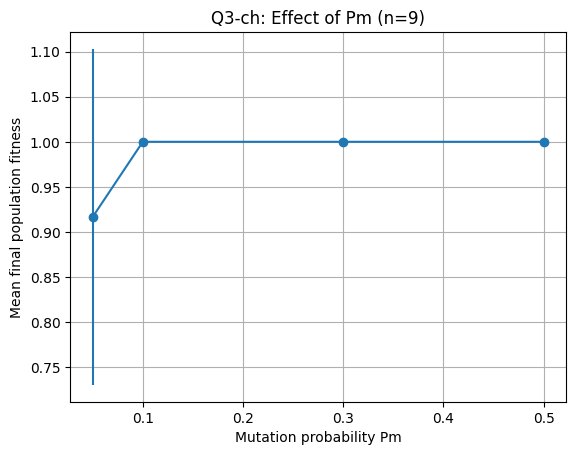
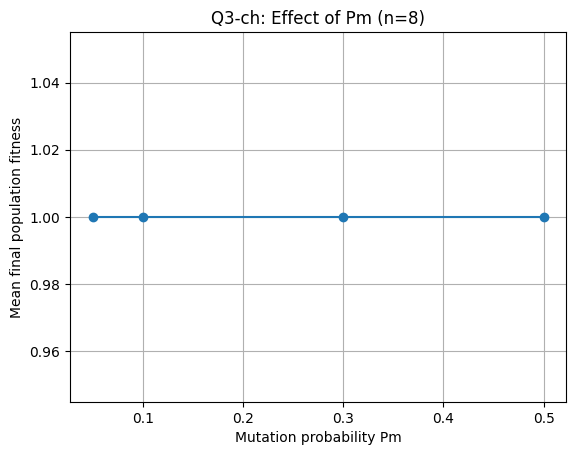
ث)



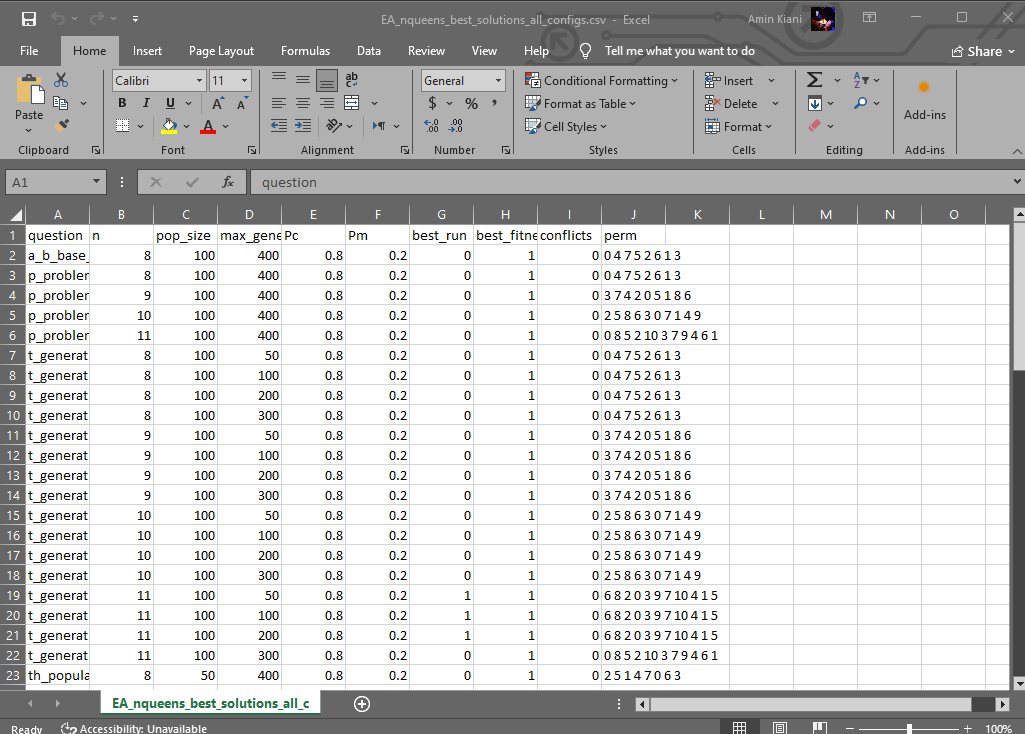
ج)



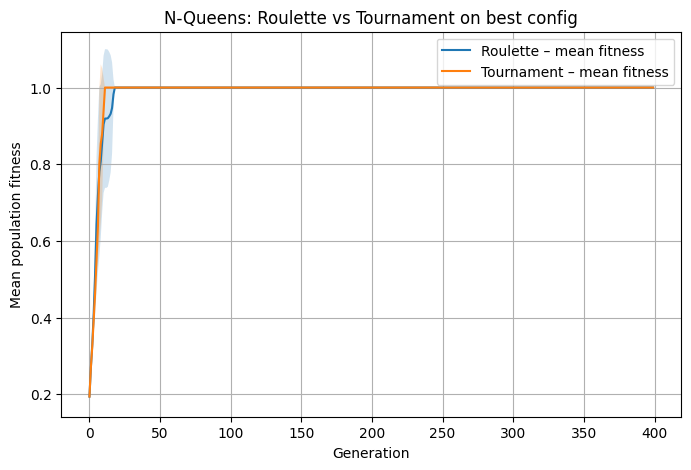
چ)



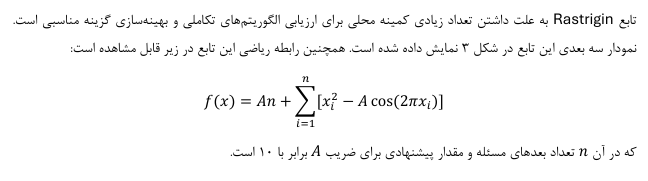
ح)

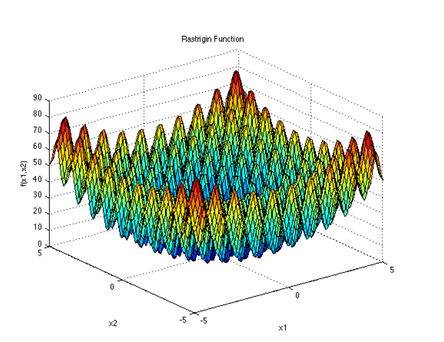


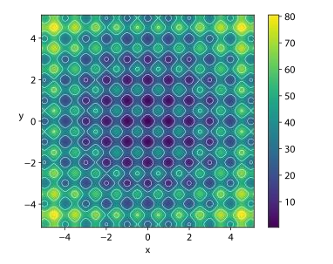
خ)



## 5- بخش 4: مسئله بهینه‌سازی تابع Rastrigin با نمایش اعشاری







نمایش: بردار اعشاری طول n (در این تمرین n=8) با مقادیر در بازه‌ی محدود بالا.

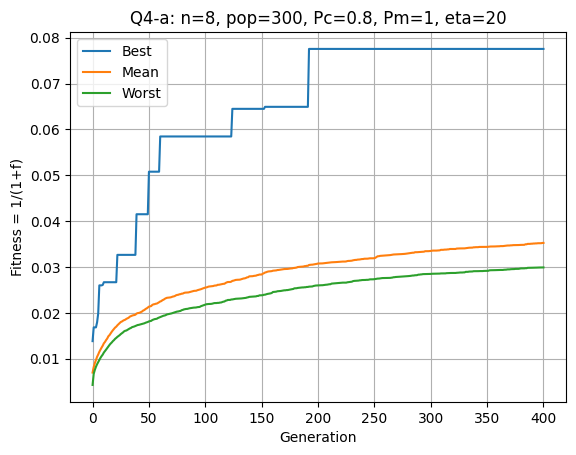
**عملگرها**

* بازترکیب: **Blend Crossover** (BLX-α) یا نسخه‌ی ساده‌ی آن، که هر ژن offspring را در بازه‌ی بین والدین (یا کمی فراتر) نمونه‌گیری می‌کند.
* جهش: **جهش چندجمله‌ای (polynomial mutation)** با پارامتر η؛
  + η کوچک یعنی جهش‌های بزرگ (اکتشاف زیاد)
  + η بزرگ یعنی جهش‌های کوچک یا بهره‌برداری local
* انتخاب: متناسب با برازندگی (roulette) و در حالت اختیاری tournament .

الف)

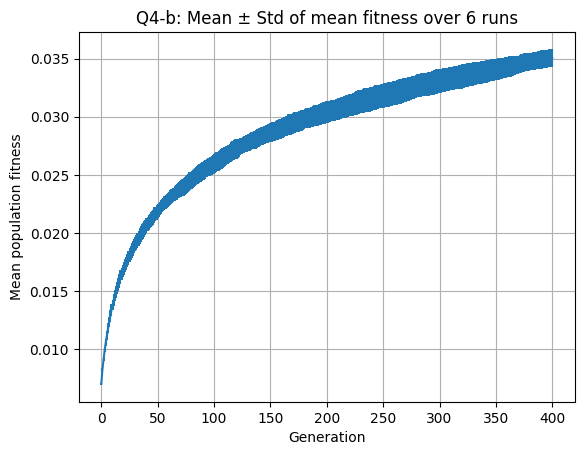
نتایج :

* در تنظیم پایه‌ی سؤال (pop=300, Pc=0.8, Pm=1, η=20)
* منحنی Best به صورت پله‌ای و کند افزایش یافت؛
* Mean و Worst آرام‌آرام رشد کردند؛



ب)

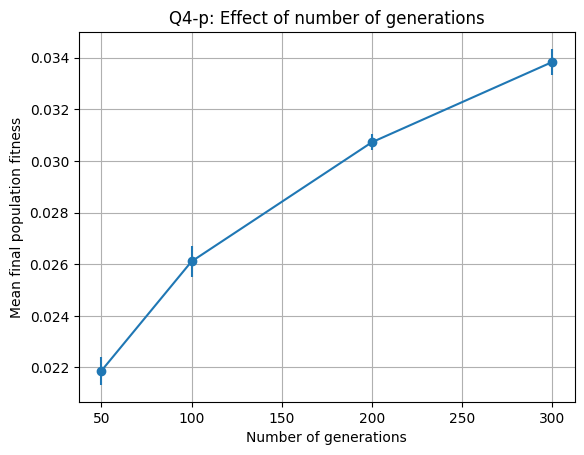
در ۶ اجرای مستقل، میانگین فیتنس نهایی نسبتاً نزدیک هم بود، اما فاصله تا بهینه‌ی واقعی هنوز قابل توجه بود. این رفتار مطابق انتظار از تابع بسیار غیرمحدب و پر از مینیمم محلی است.



پ)

تعداد نسل‌ها (۵۰، ۱۰۰، ۲۰۰، ۳۰۰):

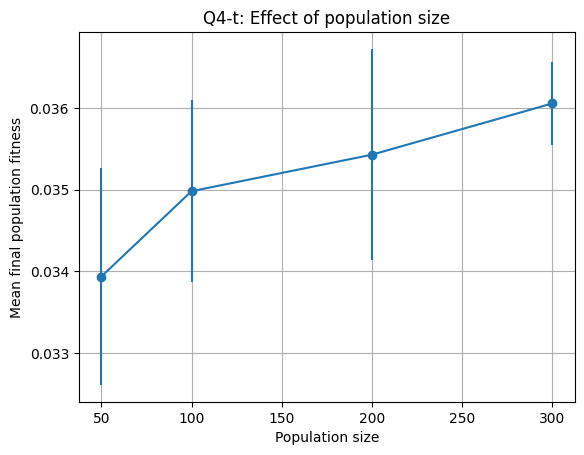
* افزایش نسل باعث بهبود steady but slow می‌شود؛
* اما به‌دلیل گیر کردن در مینیمم‌های محلی، حتی ۳۰۰–۴۰۰ نسل هم تضمین‌کننده‌ی رسیدن به f≈0 نیست؛
* از دید *چشم‌انداز برازندگی*، الگوریتم بیشتر در چند valley local دور مینیمم سراسری می‌چرخد.



ت)

اندازه‌ی جمعیت:

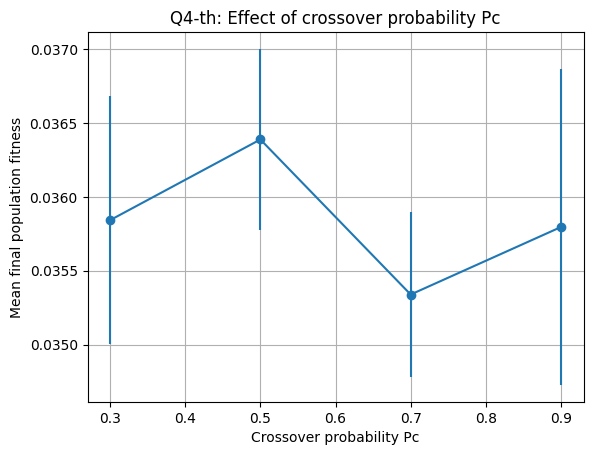
* جمعیت‌های کوچک تنوع کافی برای پوشش فضای [-5.12,5.12]^8 را ندارند و اغلب در یک محدوده‌ی کوچک گیر می‌کنند؛
* جمعیت‌های بزرگ‌تر (۲۰۰ یا ۳۰۰) توزیع اولیه‌ی بهتر و شانس بیشتری برای پوشش چند basin مختلف ایجاد می‌کنند، هرچند هزینه‌ی محاسباتی بالا می‌رود.



ث)

احتمال بازترکیب Pc :

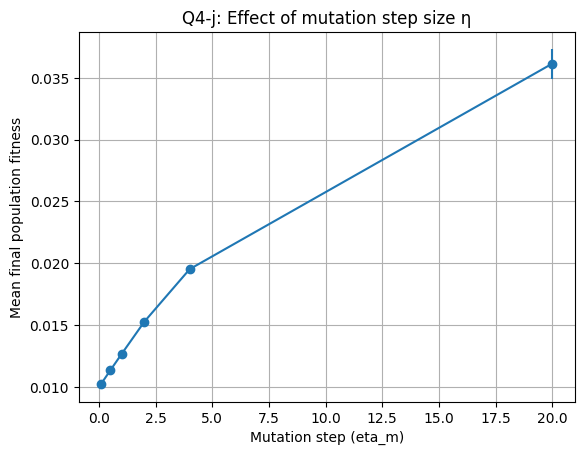
* Pc پایین یعنی offspringها به والدین خیلی شبیه می‌مانند یا اصلاً از والدین فقط کپی می‌شوند؛ سرعت جستجو کم می‌شود؛
* Pc بالا یعنی ترکیب قوی اطلاعات والدین، اما اگر جمعیت همگرا شده باشد، باز هم تأثیر کمی دارد.



ج)

گام جهش (η) :

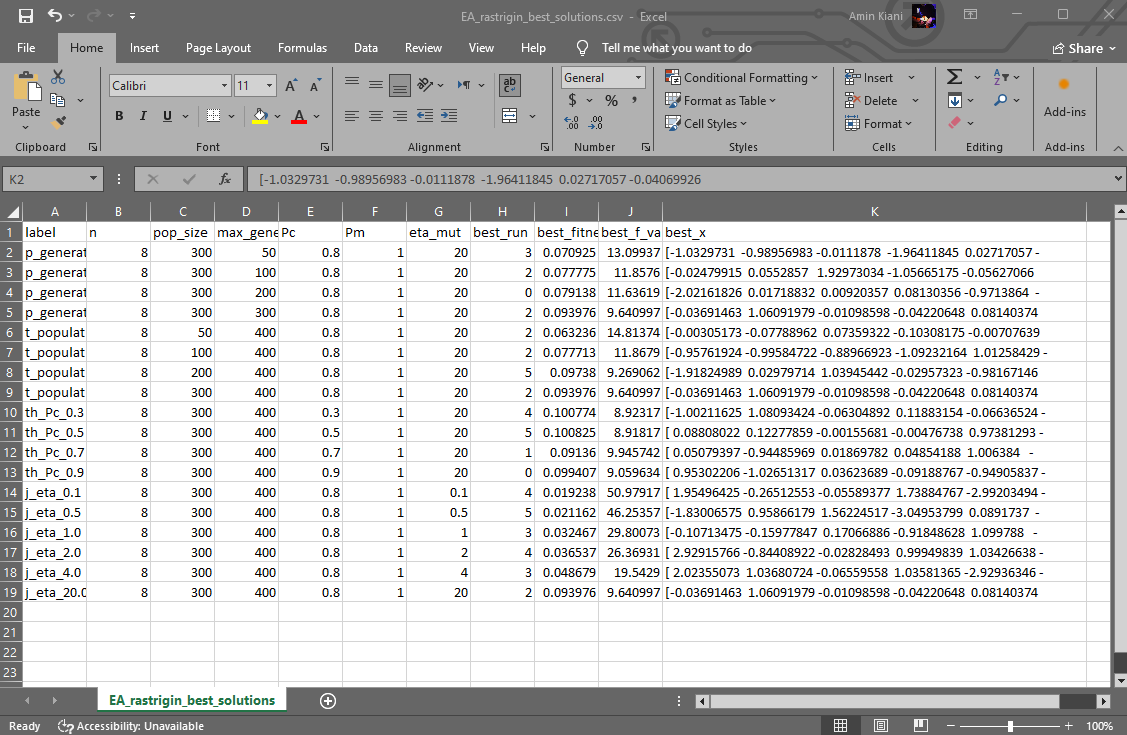
* η بسیار کوچک (مثلاً ۰٫۱، ۰٫۵) یعنی جهش‌های بزرگ یعنی جستجوی global قوی ولی سختی در fine-tuning نزدیک مینیمم؛
* η میانی(۱–۴) trade-off مناسبی بین اکتشاف و بهره‌برداری می‌دهد؛
* η بسیار بزرگ (۲۰) یعنی جهش‌های خیلی کوچک یعنی الگوریتم بیشتر شبیه local search در یک valley خاص می‌شود و از مینیمم‌های محلی عمیق بیرون نمی‌آید؛ این دقیقاً همان چیزی است که در نمودار Q4-a است.



چ)

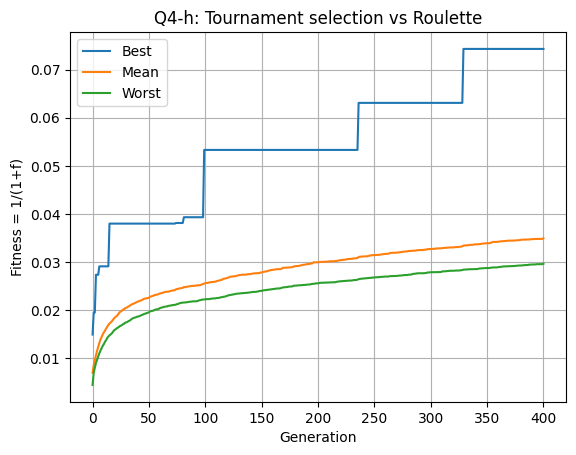
**بهترین راه‌حل‌ها**

* برای هر تنظیم، بهترین بردار x∗ همراه با مقدار f(x∗) در csv ثبت شده است؛
* مقایسه تنظیم‌های مختلف نشان داد:
  + تنظیم‌هایی با جمعیت بزرگ‌تر، Pc بالا و η میانی معمولاً f پایین‌تری تولید می‌کنند؛



ح)

انتخاب tournament برای تنظیم‌های خوب باعث همگرایی سریع‌تر ولی گاهی گیر کردن بیشتر در local minima می‌شود، در حالی‌که roulette فشار انتخاب نرم‌تری دارد و تنوع را بیشتر حفظ می‌کند.



## 6- مراجع

[https://github.com](https://github.com/M-Amin-Kiani)

https://stackoverflow.com/questions

[https://colab.research.google.com/](https://colab.research.google.com/%20o)

https://algorithmafternoon.com/books/genetic\_algorithm/chapter04/

https://chatgpt.com/

https://grok.com/