



دانشگاه اصفهان

دانشکده مهندسی کامپیوتر - گروه هوش مصنوعی و رباتیک

گزارش تمرین اول - فرآیندهای تکاملی

EA

```
Algorithm: EA
1   pop = Generate popSize initial candidate solutions of size problemSize
2   popFit = Evaluate pop using f(x)
3   While not Terminate():
4       parentsPool = Select popSize solutions from pop using popFit
5       parentPairs = Shuffle parentsPool and randomly Pair solutions
6       offspr = Perform Recombination on parentPairs with P_c
7       offspr = Perform Mutation on offspr with P_m
8       offsprFit = Evaluate offspr using f(x)
9       [pop, popFit] = Select best popSize solutions from [pop + offspr] using popFit and offsprFit
10  Return best solution in pop
```

پدیدآورنده:

محمد امین کیانی

۴۰۴۳۶۴۴۰۸

دانشجوی کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد درس: دکتر کارشناس

نیمسال اول تحصیلی ۱۴۰۴-۰۵

فهرست مطالب

۳	مستندات.....
۳	۱- مسئله و تحلیل کلی آن:.....
۷	۲- بخش ۱: مسئله کوله‌پشتی با نمایش دودویی
۴۸	۳- بخش ۲: مسئله گروه‌بندی اعداد با نمایش عددی.....
۶۲	۴- بخش ۳: مسئله چند وزیر با نمایش جایگشت
۷۵	۵- بخش ۴: مسئله بهینه‌سازیتابع Rastrigin با نمایش اعشاری
۸۴	۶- مراجع.....

مستندات

۱- مسئله و تحلیل کلی آن:

در اینجا یک الگوریتم ژنتیک داریم که شامل تولید جمعیت اولیه، ارزیابی برازنده‌گی، 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 ۱ تا ۴، 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 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 P_c 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 خیلی بالا distribution bias Pm 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 گستته مناسب است.

- تنوع جمعیت و همگرایی: تنوع در نسل‌های اولیه بالا است.

توضیح اضافی نحوه انجام تمرین:

تمرین با پیاده‌سازی یک چارچوب EA ماثولار در Python انجام شد (کد کامل اسکرچ قابل اجرا در گوگل کولب). سپس هر بخش جداگانه تست شد و نتایج تصادفی هستند، اما ثابت برای تکرارپذیری استفاده شده‌است. این رویکرد درک جامعی از تأثیر پارامترها بر اساس landscape مسئله می‌دهد و EA را برای مسائل متتنوع 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 به کار 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.stdev و statistics.mean ○

برای محاسبه‌ی میانگین و انحراف معیار برازنده‌گی در چند اجرای مستقل هر تنظیم پارامتر. این مقادیر برای رسم نمودارهای «میانگین \pm 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، Pm، Pc

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

۲- بخش ۱: مسئله کوله پشتی با نمایش دودویی

مودودی داده ای EA

تاریخ _____ موضوع _____

i	w_i	v_i
۱	۱	۳
۲	۰	۰
۳	۱	۲
۴	۱	۴
۵	۰	۶

$W = 10$

۱- مسئله کوله پشتی با غایض دو دویس:

(الف) با زانو تابع برآندازی؟

simple knapsack = $\sum_{i=1}^{PS} v_i u_i = (1 \times 3) + (1 \times 0) + \dots + (1 \times 6)$

$f(\vec{u}) = \begin{cases} \sum w_i v_i \leq W & \text{دیر توابع برآندازه} \\ 0 > W & \end{cases}$

جون وزن جهی آنچه با هم بازم
در شرط برقرار است یعنی:

/ حل بینه $\vec{u} = (1, 0, 1, 1, 0, 1) \Rightarrow 1+2+1+1+0 = 5 \leq 10$

? وزن جهی با هم از طرفی هنوز هم کمتر است پس استخاب جهی آنچه هام قدر رعایت می کند و هم بینی جمع ارزش میان عدد پس میں بردار برآندازه هرسه تابع بینه است.

$\sum w_i = V \leq 10$, $\sum v_i = 3+0+2+4+6 = 15$

full knapsack = $\prod_{i=1}^{PS} u_i v_i = (1 \times 3) \times (1 \times 0) \times (1 \times 2) \times (1 \times 4) \times (1 \times 6) = 1440$

stupid knapsack = $PS \times \prod_{i=1}^{PS} u_i v_i - \sum_{i=1}^{PS} v_i u_i = 5 \times 1440 - 15 = 14085$

کل برآندازه تابع برآندازه، راه حل بینه جهان بردار ناماگ با محدودیت برآندازه بالاست.

۱) زیرمسئله‌ی تئوری (۵ آیتم) – حل دستی

سوال) مطابق نوشه‌های داخل دفتر، یعنی همه‌ی آیتم‌ها با هم قابل انتخاب‌اند و هیچ کاندیدائی از نظر قید ظرفیت نامعتبر نمی‌شود. بنابراین برای هر تابع برازنده‌ی استانداردی که قید ظرفیت را رعایت کند (مثلاً تابعی که اگر وزن بیش از W شد مقدار ۰ بگیرد) یا برای نسخه‌هایی که مجازات دارند یا نسخه‌ی Full (که اساساً ظرفیت را نادیده می‌گیرد)، بهترین راه حل مسئله در این نمونه‌ی کوچک انتخاب همه‌ی آیتم‌ها خواهد بود.

- راه حل بهینه: بردار ۱

Simple = 20 | Full = 720 | Stupid = 17980

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

در این کد و بخش‌های 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):

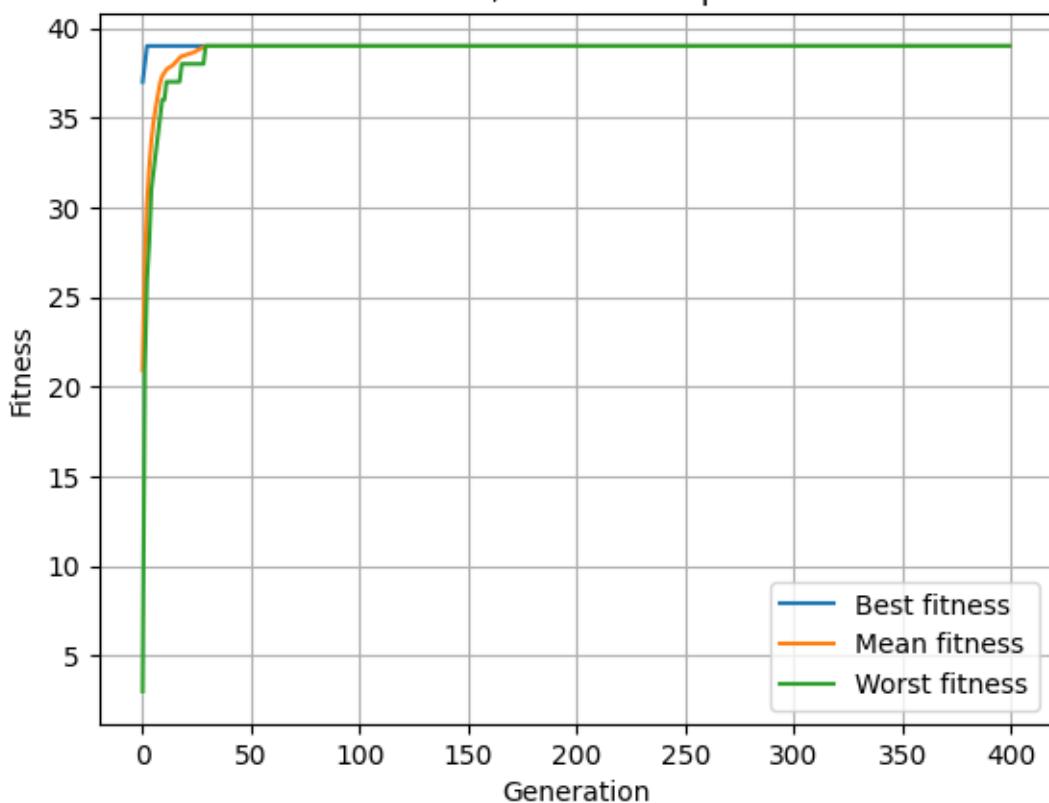
- برای Simple و Stupid قید ظرفیت را اعمال می‌کنیم
- برای Full، قید ظرفیت را نادیده می‌گیریم.(در فراخوانی run_ea_knapsack باید ignore_capacity_for_full=True باشد.)

(الف)

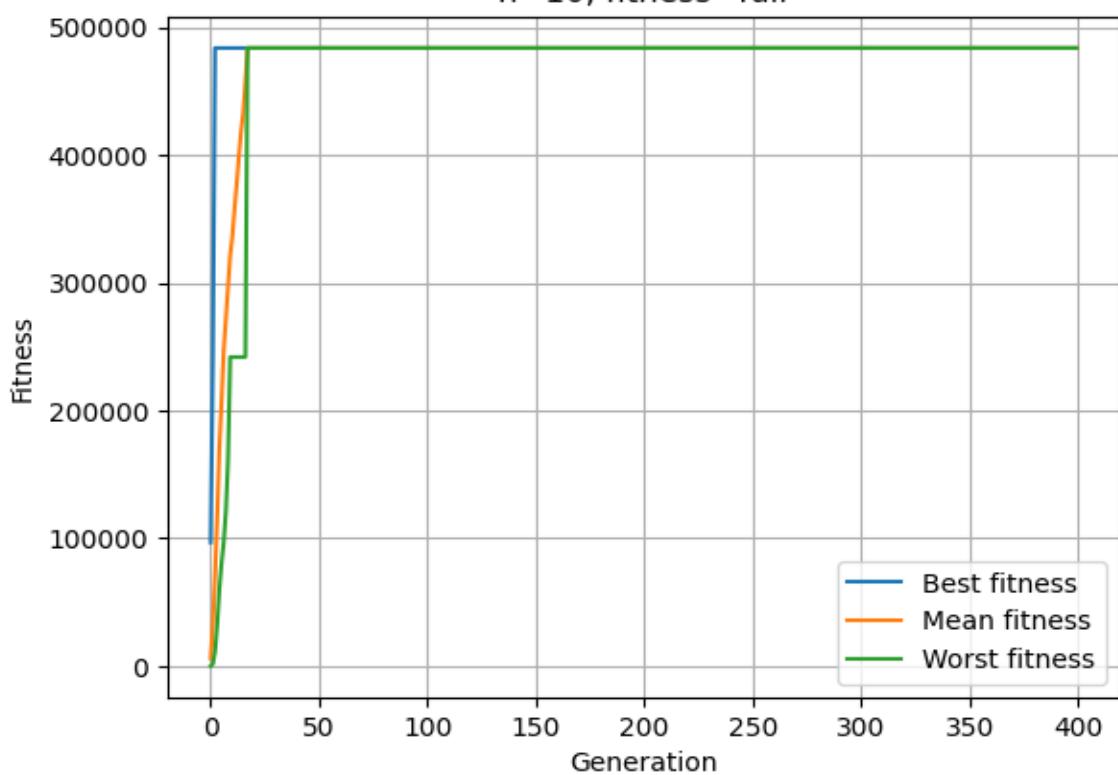
رفتار نمودارهای تکامل (Best / Mean / Worst) نشان دادند که:

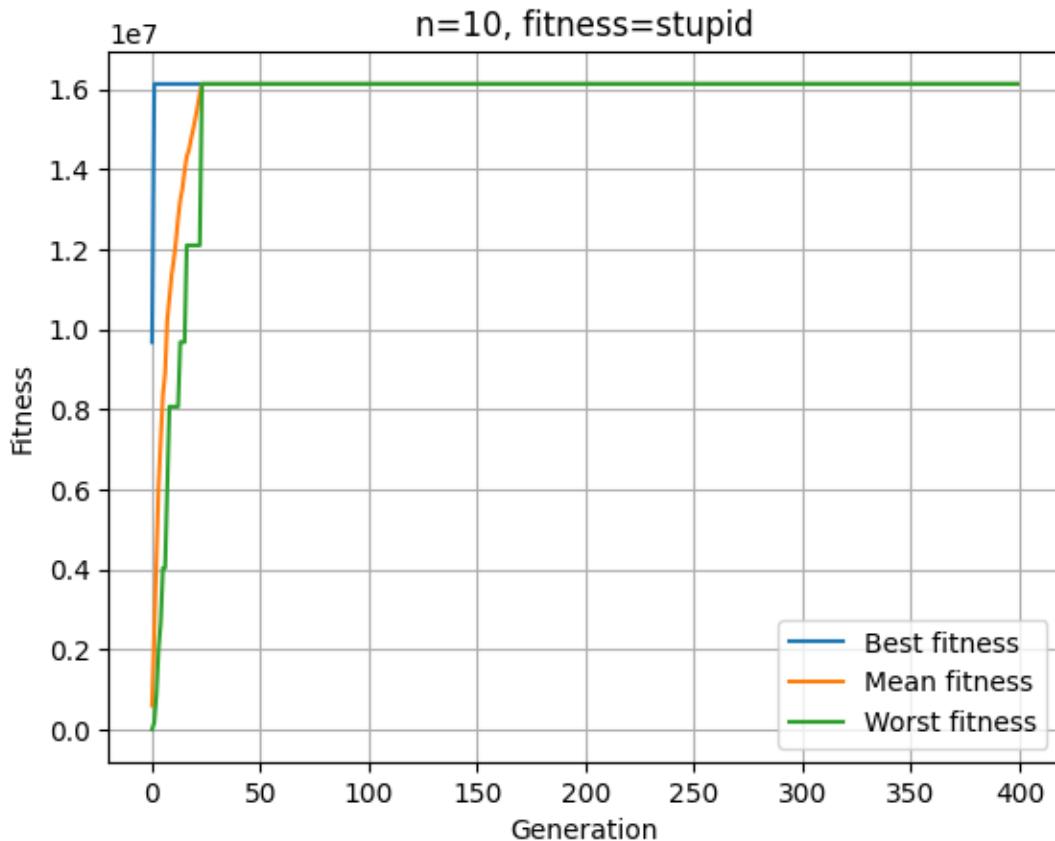
برای تابع Simple ، الگوریتم سریع‌تر به جواب‌های خوب می‌رسد، چون landscape برازنده‌ی نسبتاً صاف و یکنواخت است ولی برای Full و مخصوصاً Stupid landscape بسیار تیزتر و ناهموار است و کوچک‌ترین تغییر بیت می‌تواند برازنده‌ی را به صفر نزدیک کند؛ در نتیجه، همگرایی کندر و پله‌ای تر است.

$n=10$, fitness=simple



$n=10$, fitness=full



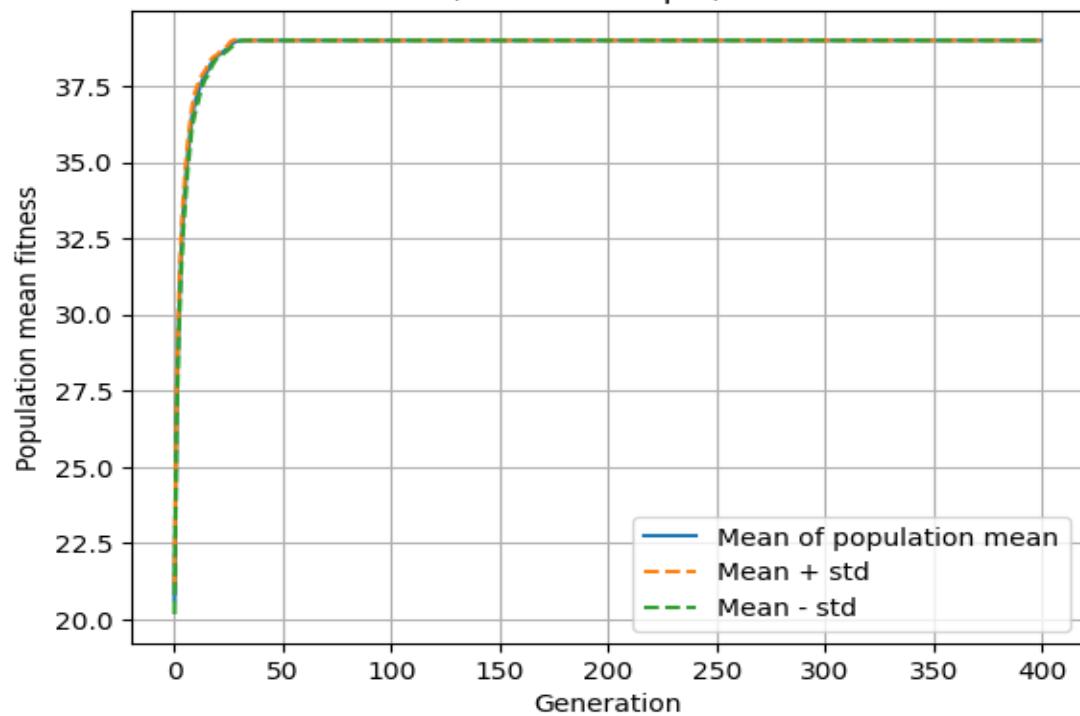


(ب)

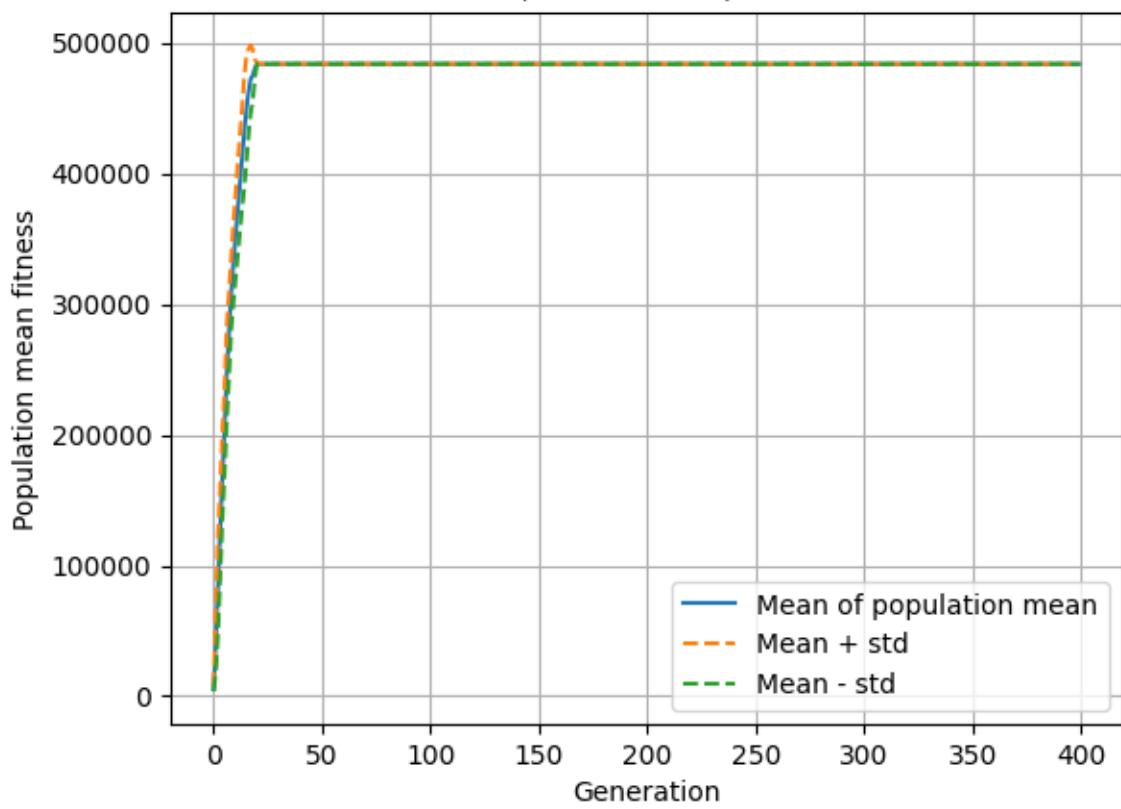
در چند اجرای مستقل (حداقل ۶ بار و تست ۱۲ بار) مشاهده شد که:

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

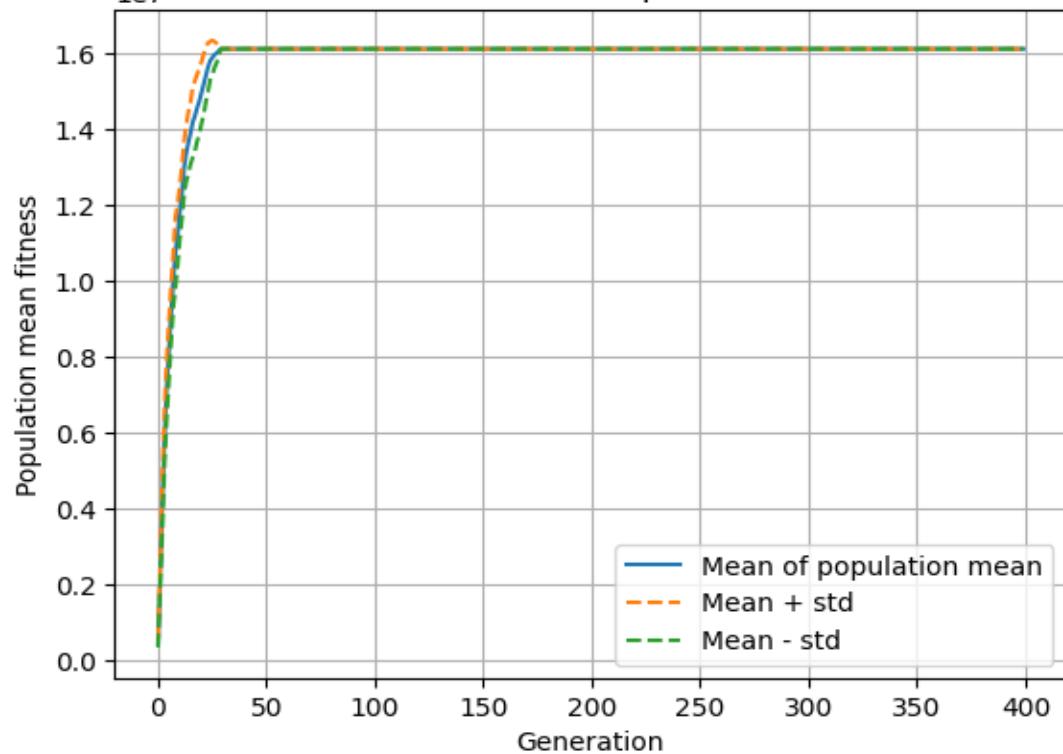
$n=10$, fitness=simple, runs=6



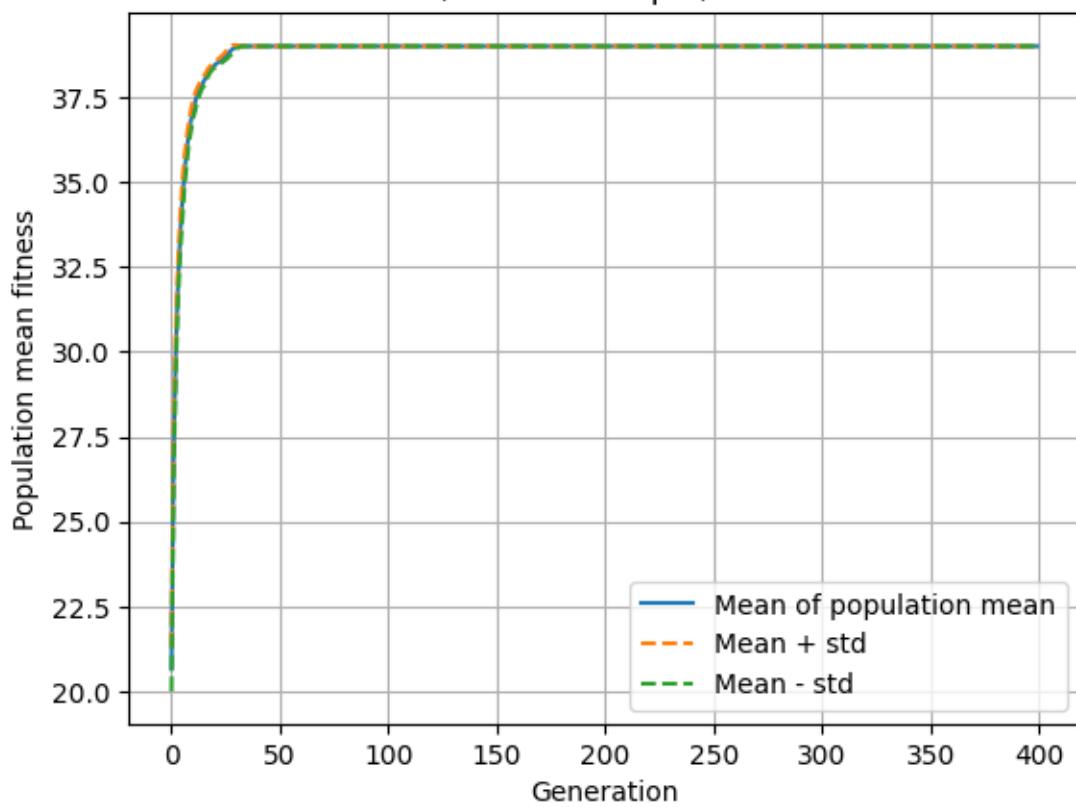
$n=10$, fitness=full, runs=6



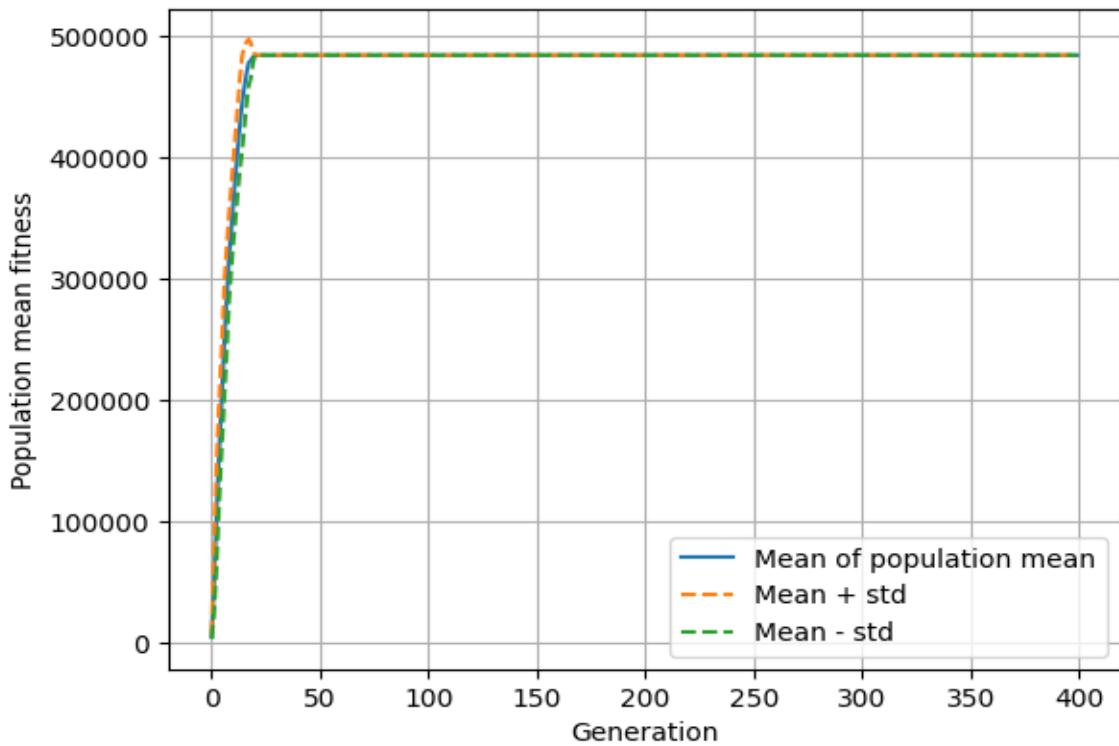
$n=10$, fitness=stupid, runs=6



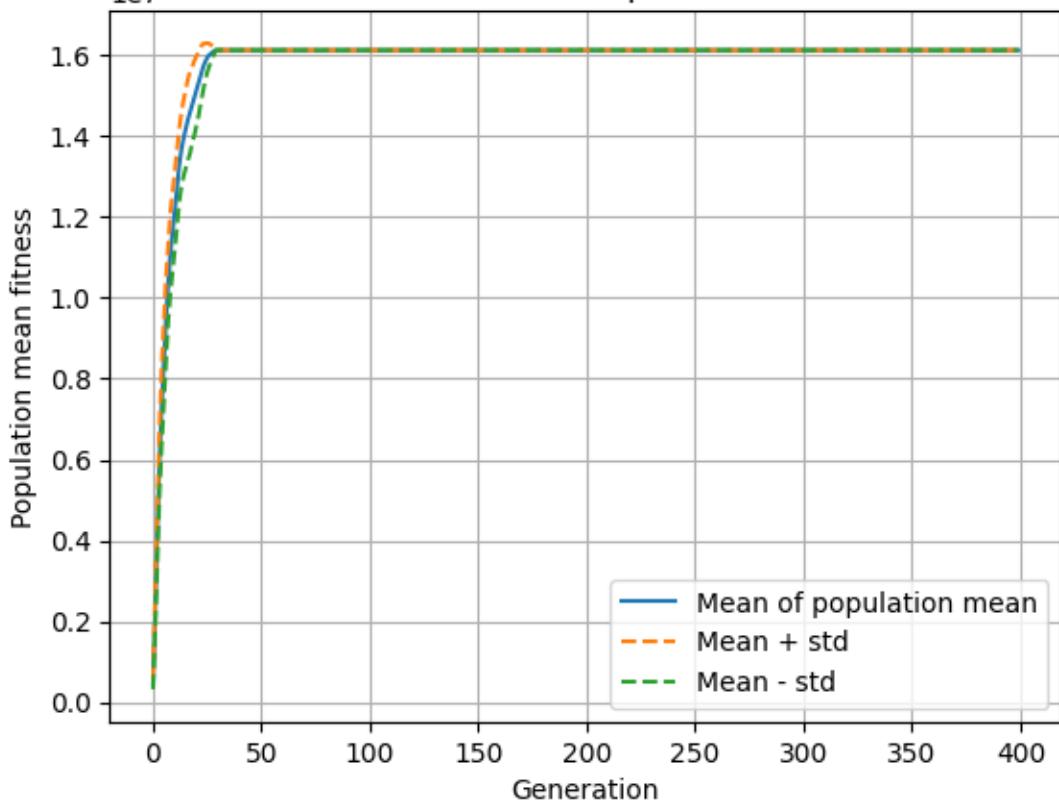
$n=10$, fitness=simple, runs=12



$n=10$, fitness=full, runs=12



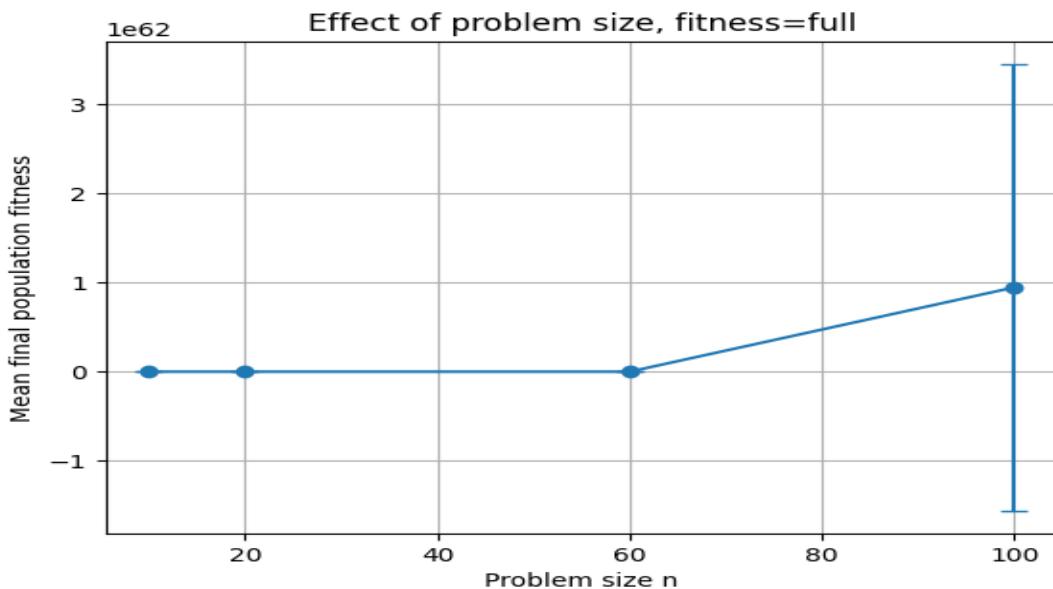
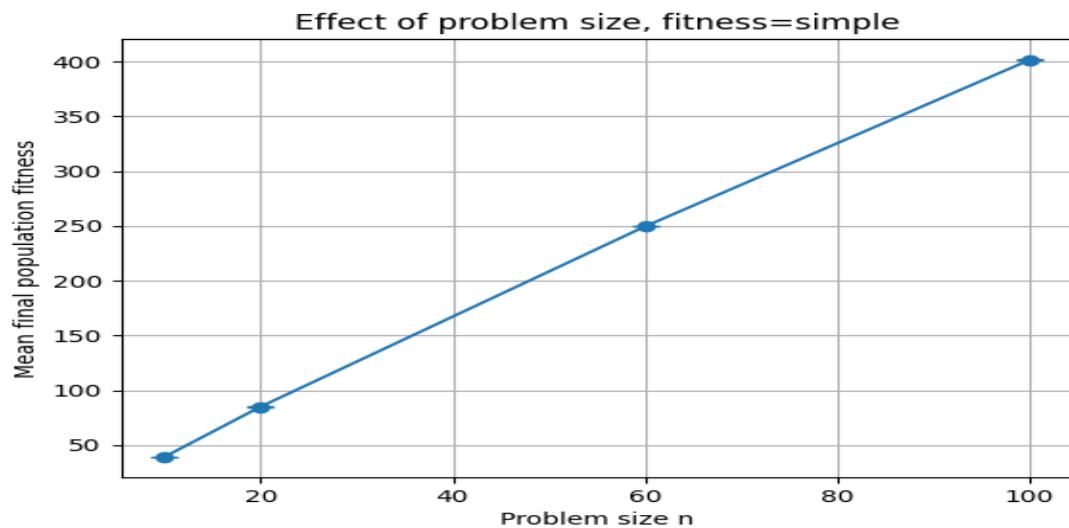
$n=10$, fitness=stupid, runs=12

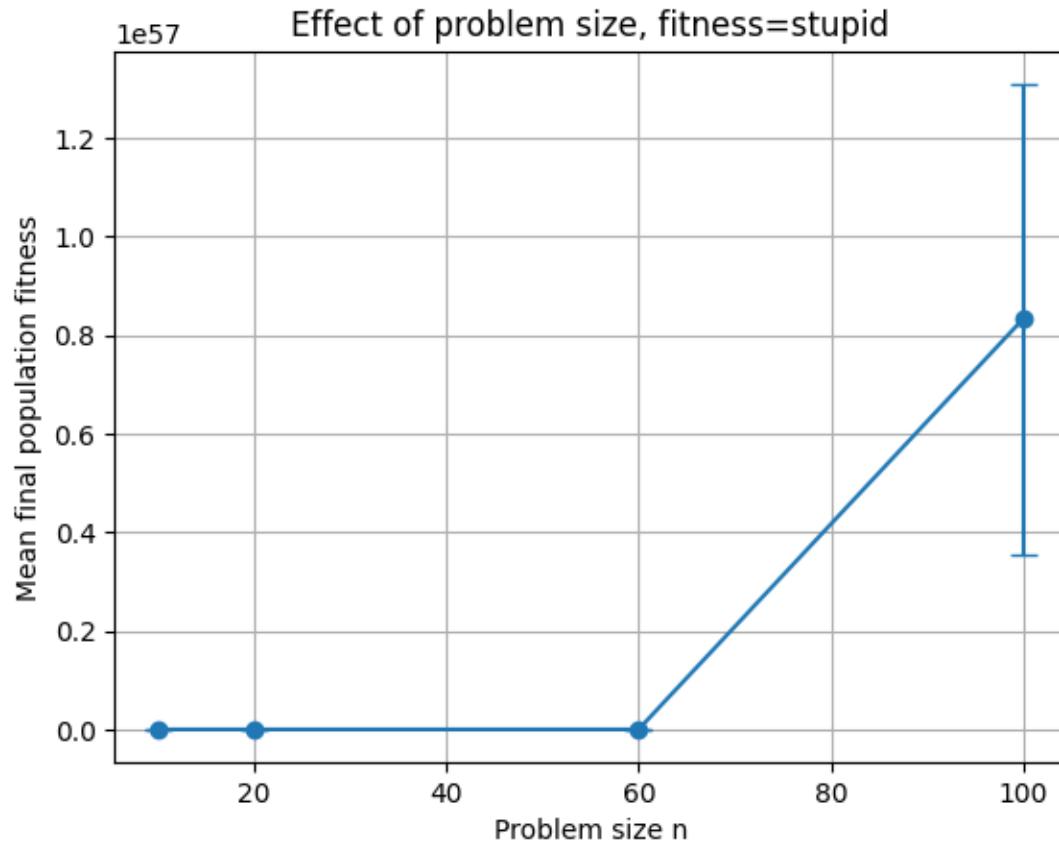


(پ)

افزایش n (۱۰، ۲۰، ۶۰، ۱۰۰):

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



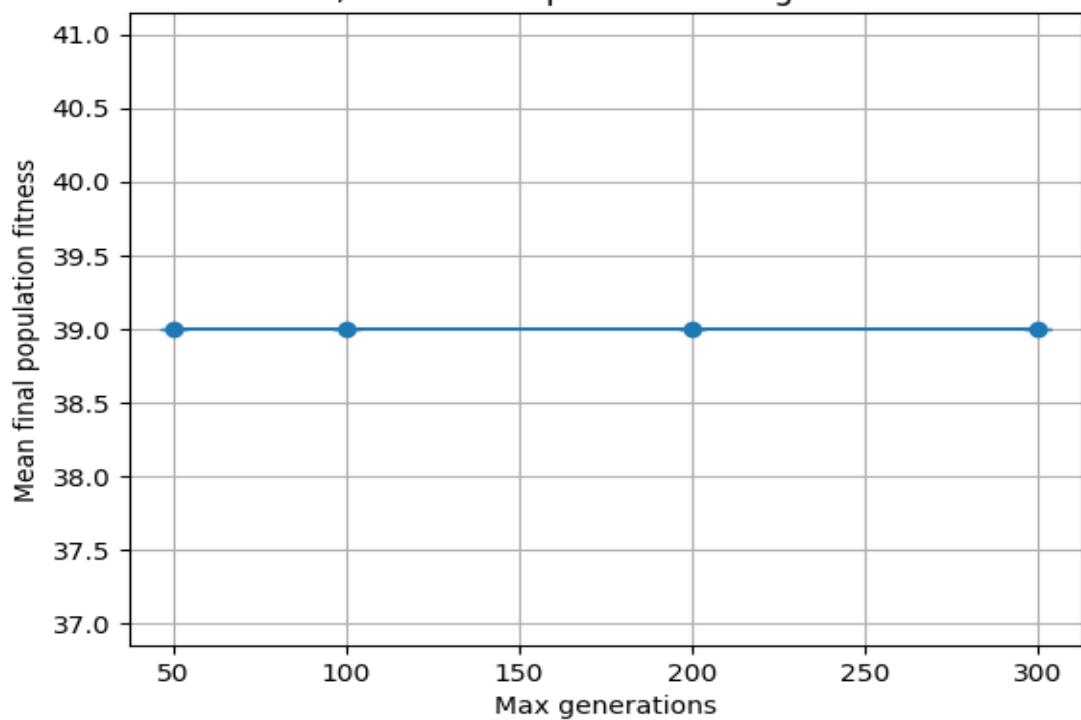


(ت)

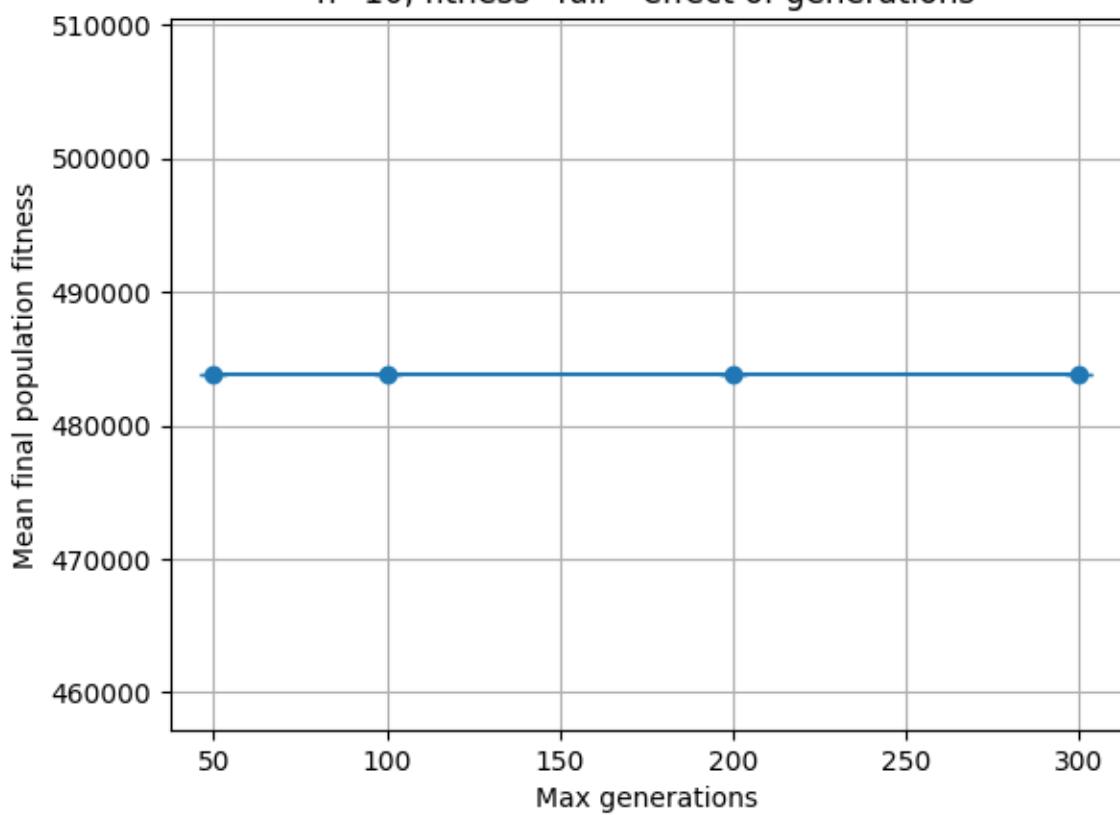
افزایش تعداد نسل‌ها (۵۰، ۱۰۰، ۲۰۰، ۳۰۰، ۴۰۰):

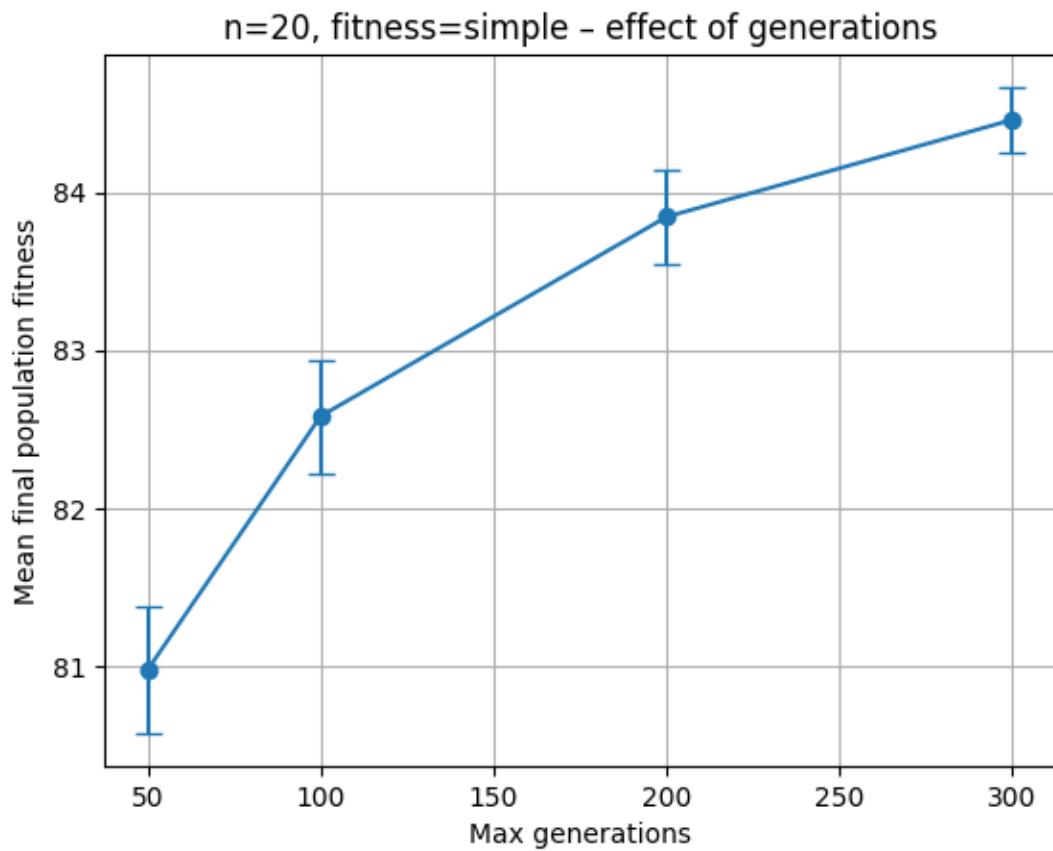
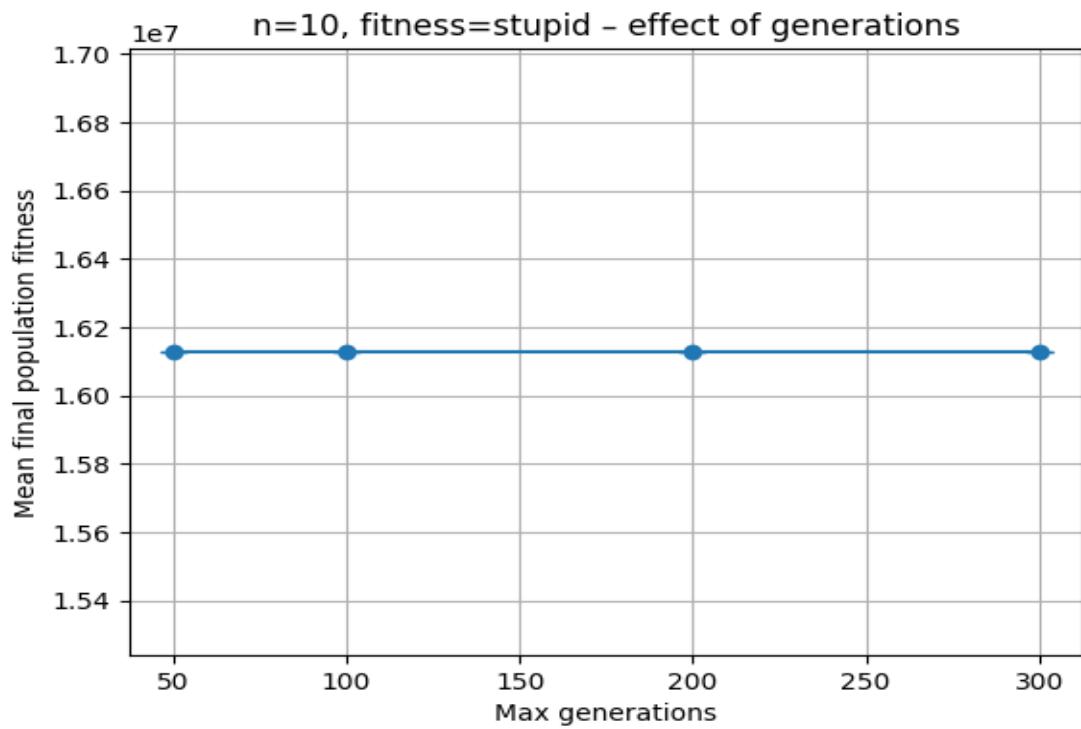
- در نسل‌های کم عملکرد ضعیف است؛
- بعد از یک حد مشخص، افزایش نسل تنها بهبود کوچک (diminishing returns) ایجاد می‌کند؛
- از دید فشار انتخاب، تعداد نسل بیشتر فرصت بیشتری برای انتشار ژن‌های خوب می‌دهد، ولی اگر تنوع جمعیت کم شده باشد، حتی نسل‌های زیاد هم کمکی نمی‌کند.

n=10, fitness=simple - effect of generations

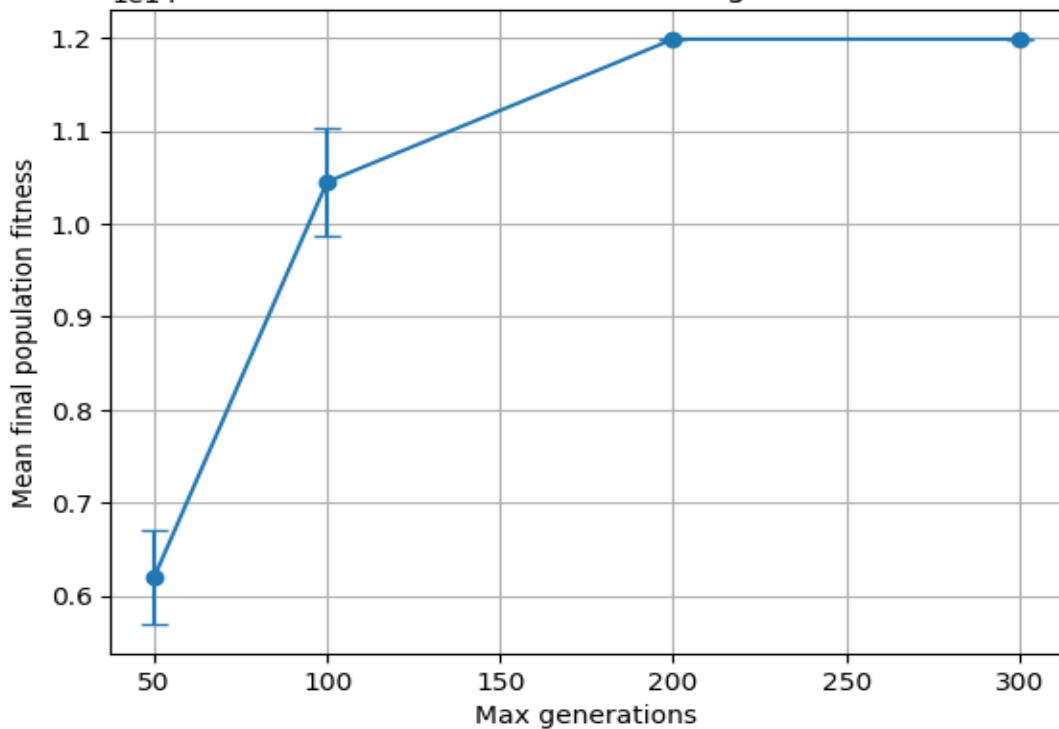


n=10, fitness=full - effect of generations

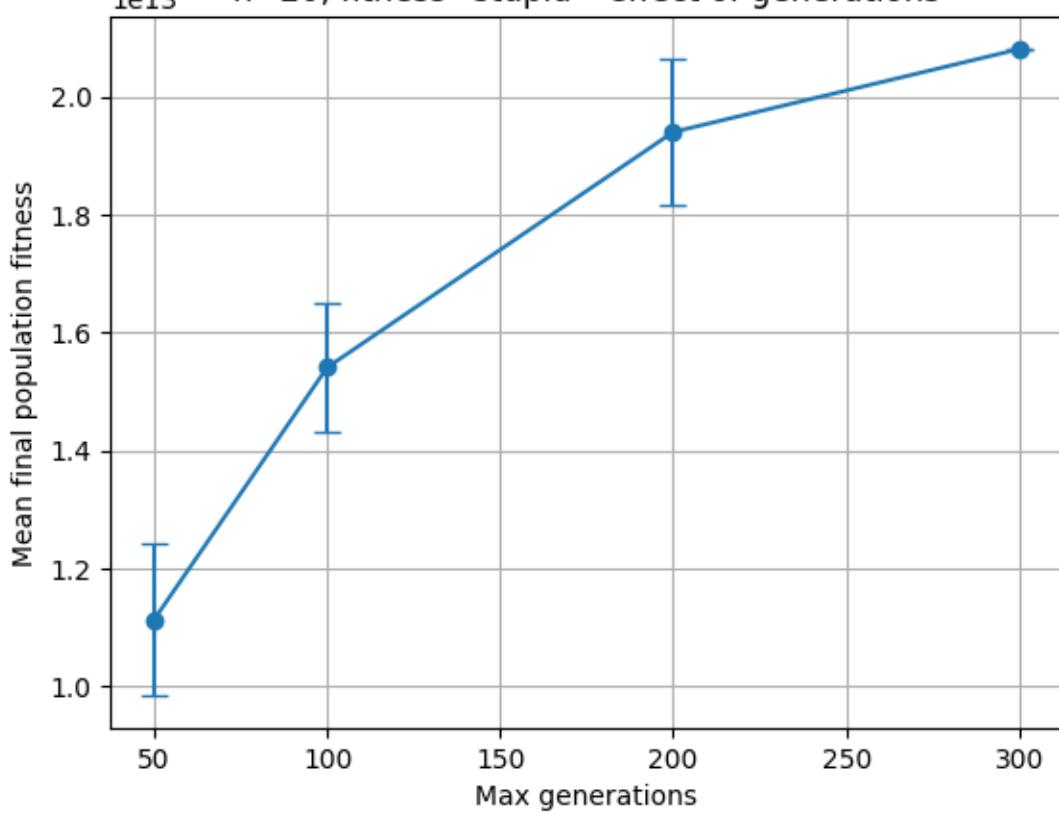




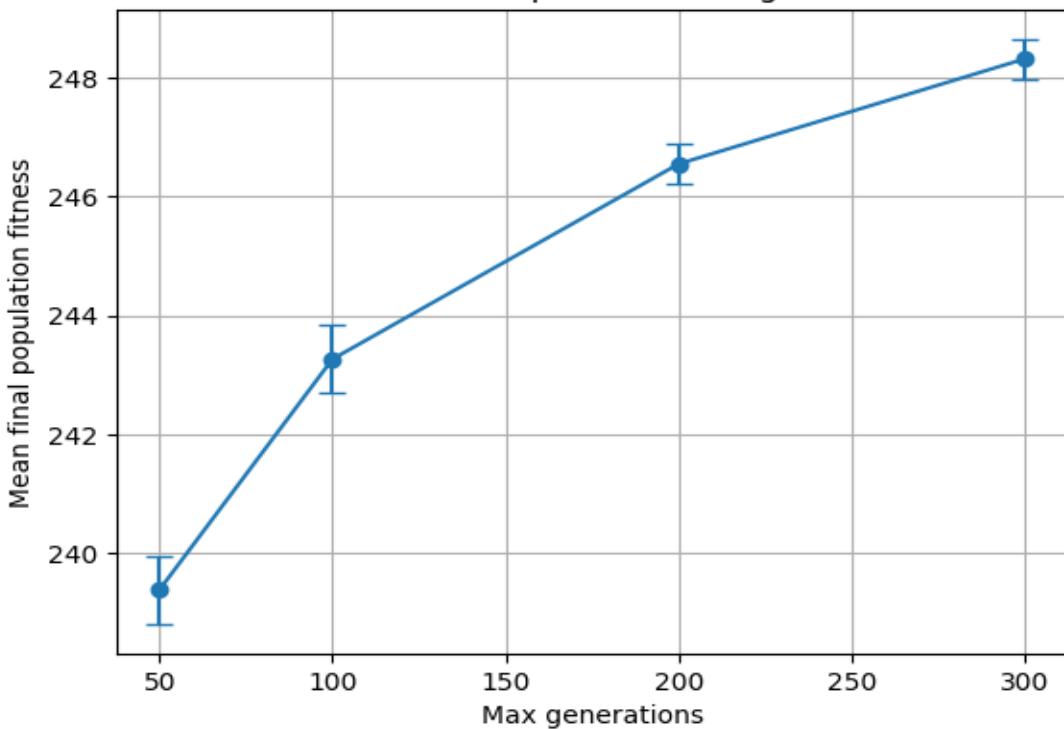
$n=20$, fitness=full - effect of generations



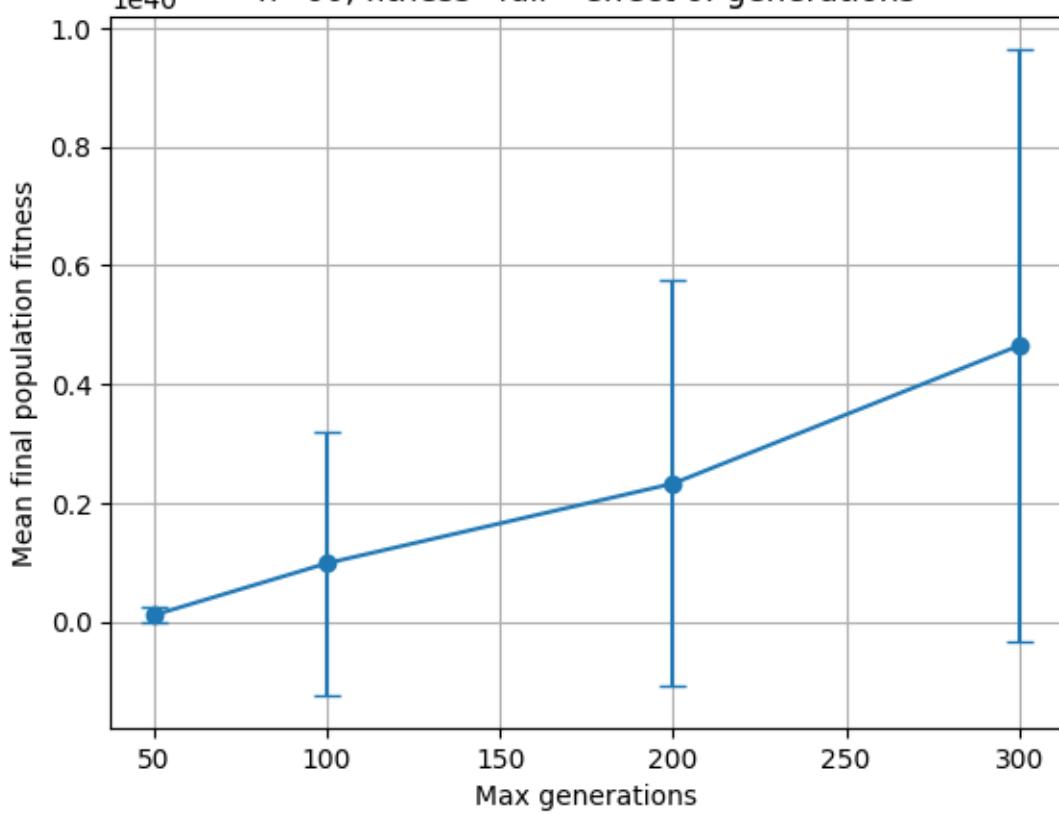
$n=20$, fitness=stupid - effect of generations



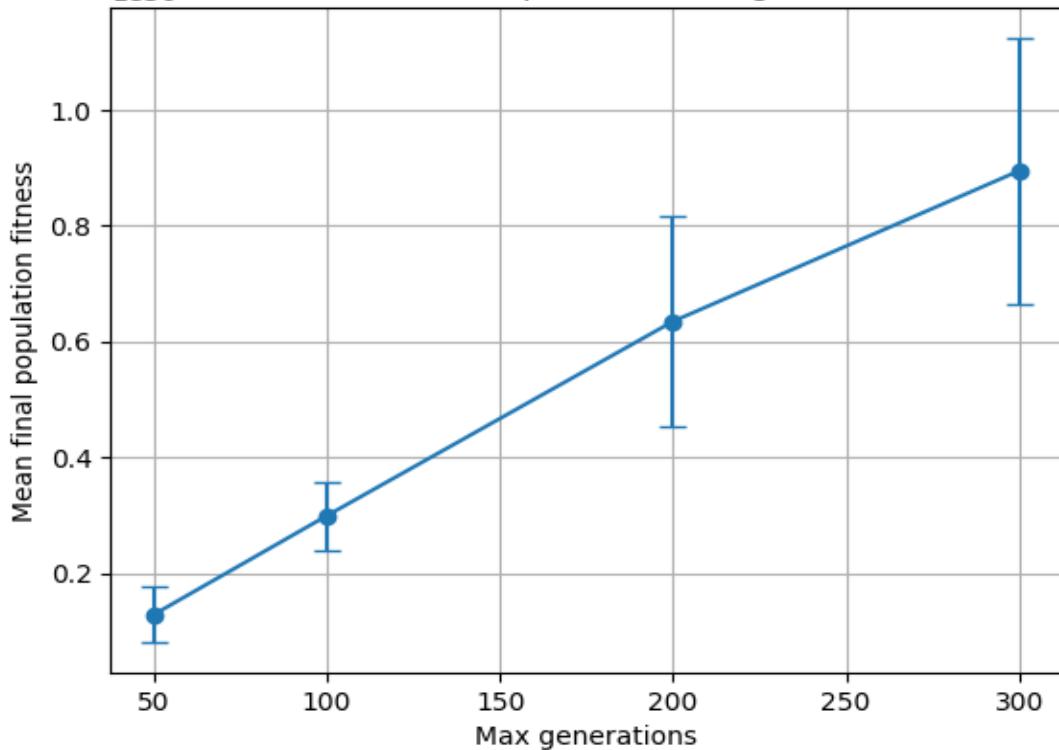
n=60, fitness=simple - effect of generations



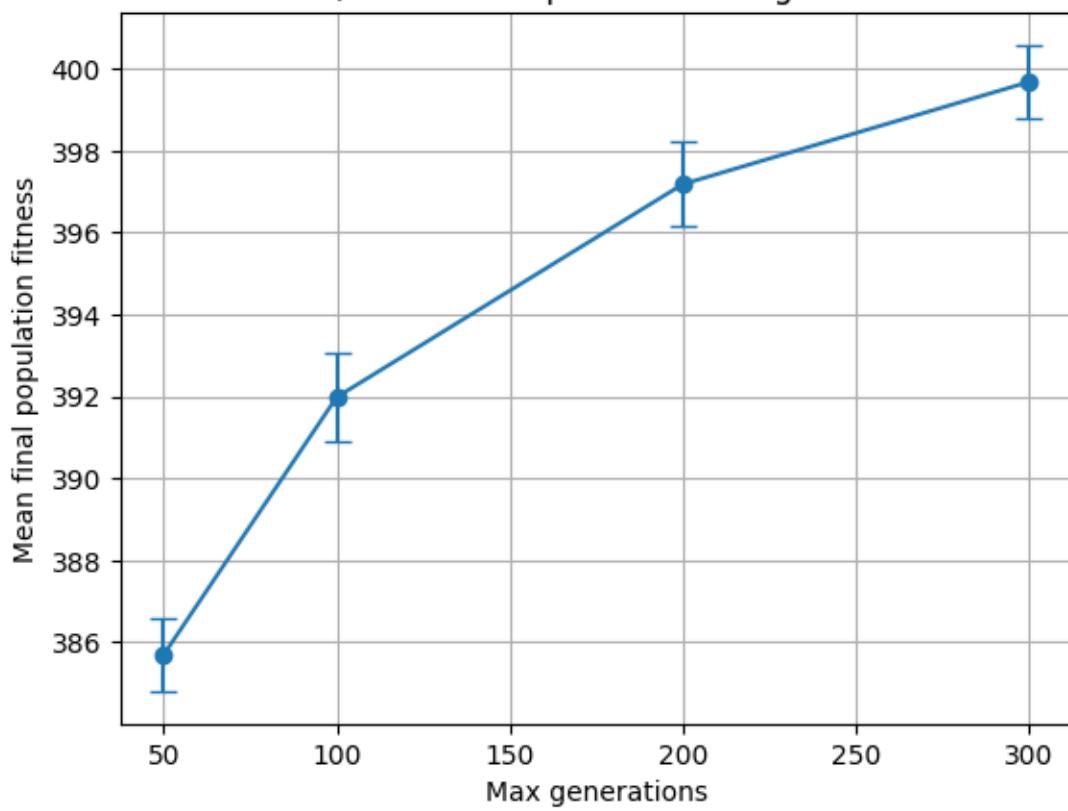
n=60, fitness=full - effect of generations



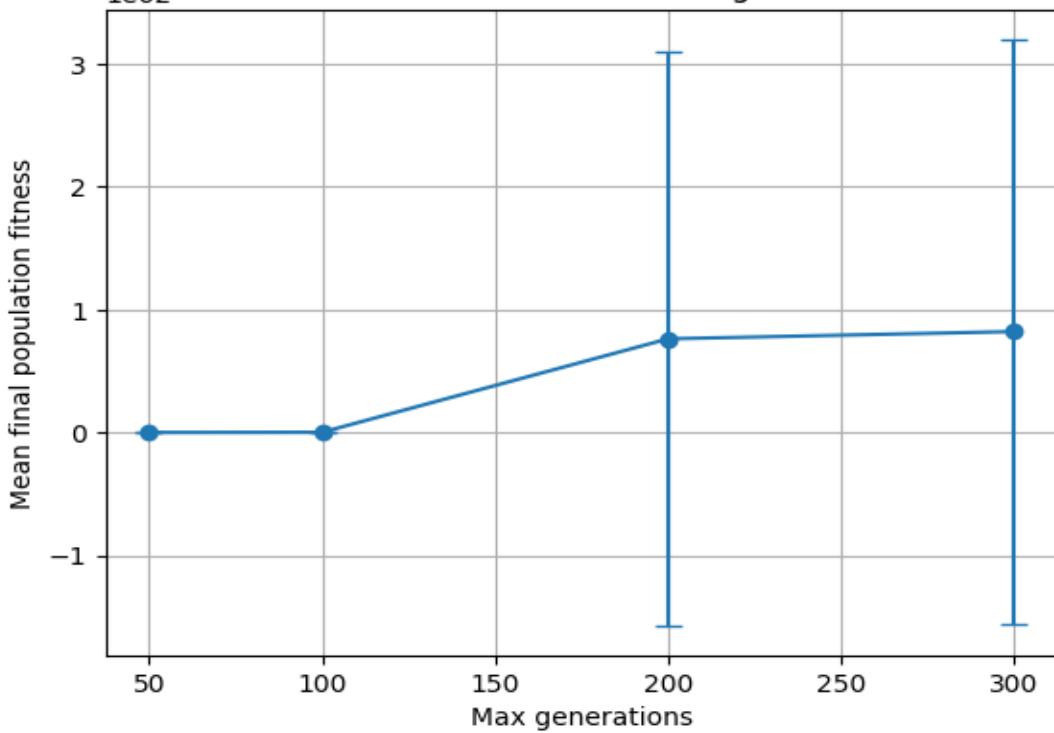
1e36 n=60, fitness=stupid - effect of generations



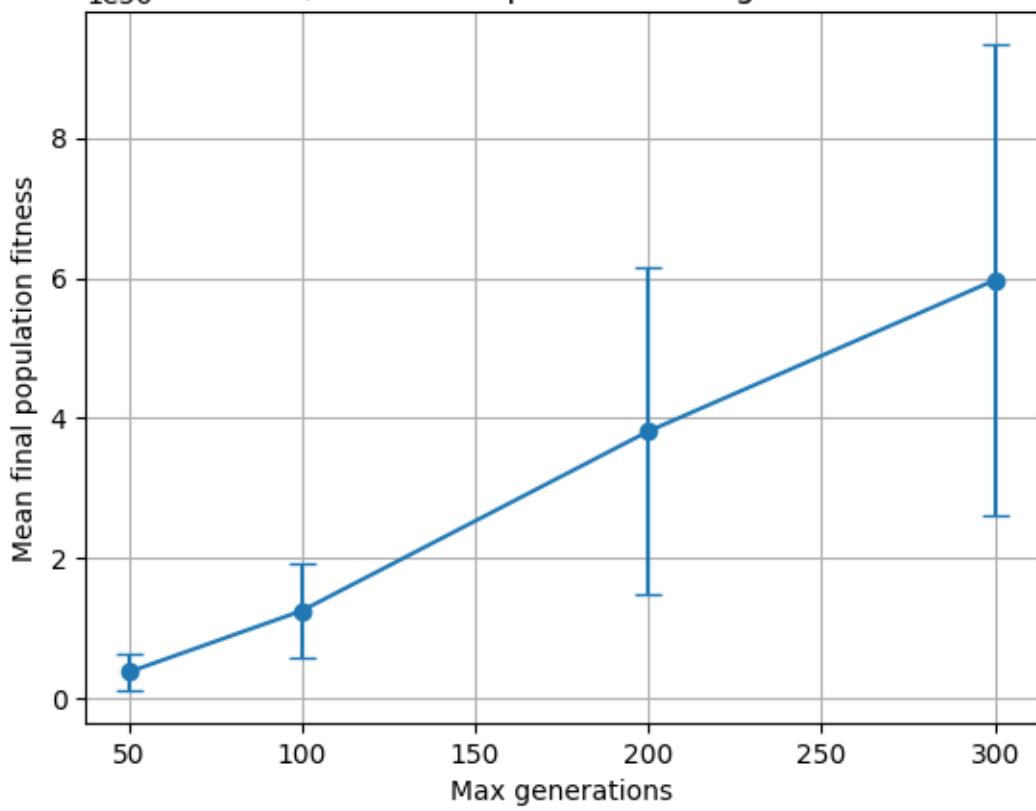
n=100, fitness=simple - effect of generations



$1e62$ $n=100$, fitness=full - effect of generations



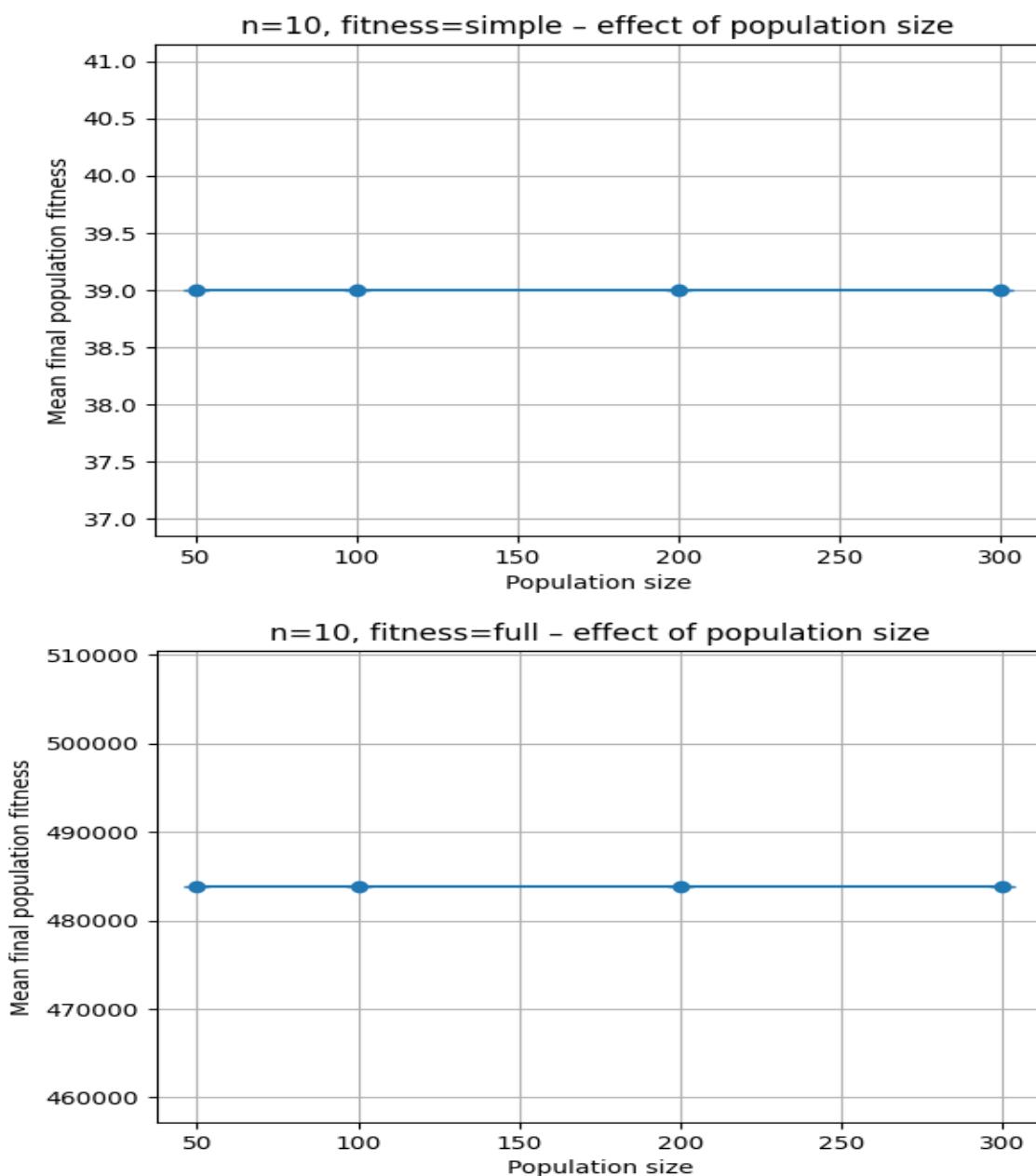
$1e56$ $n=100$, fitness=stupid - effect of generations

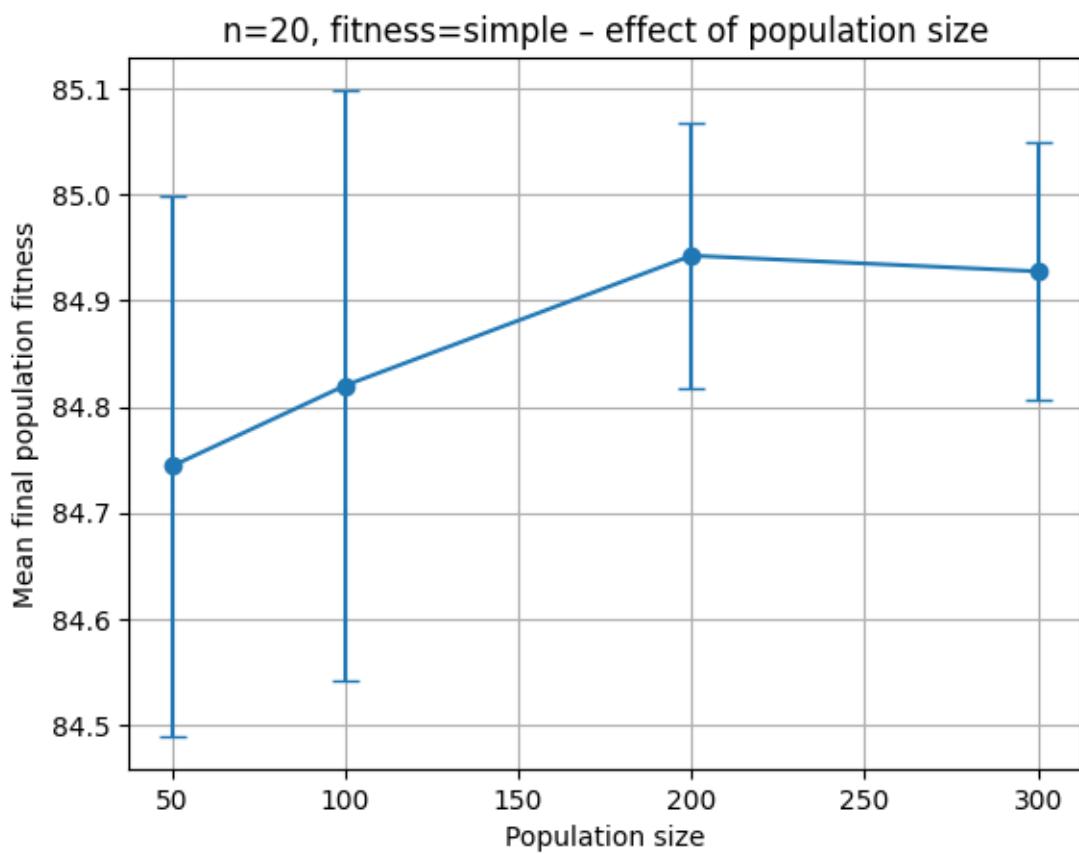
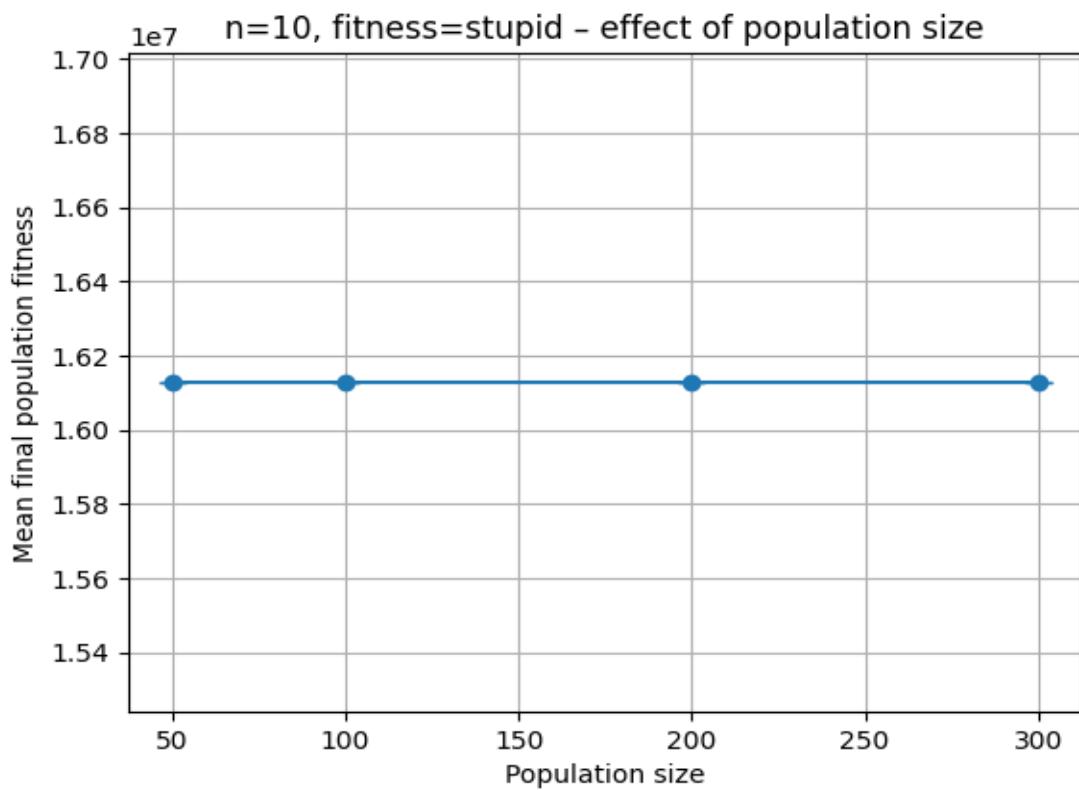


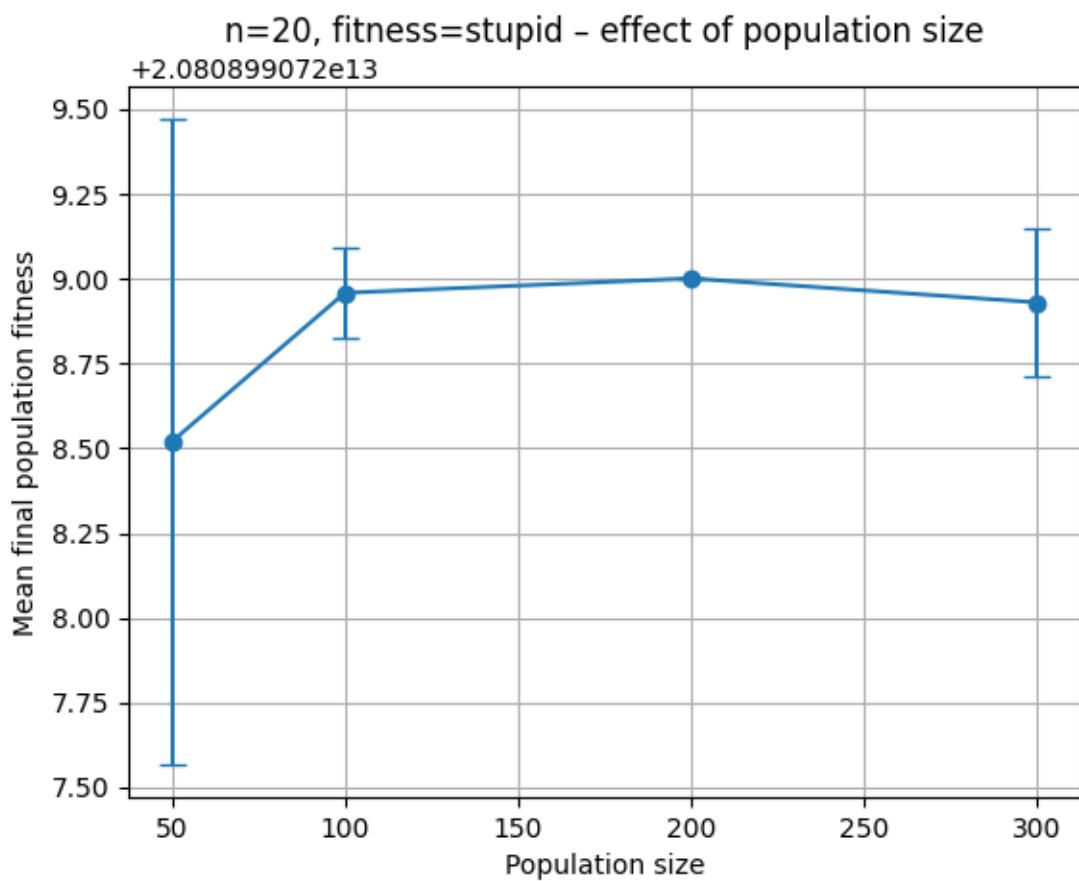
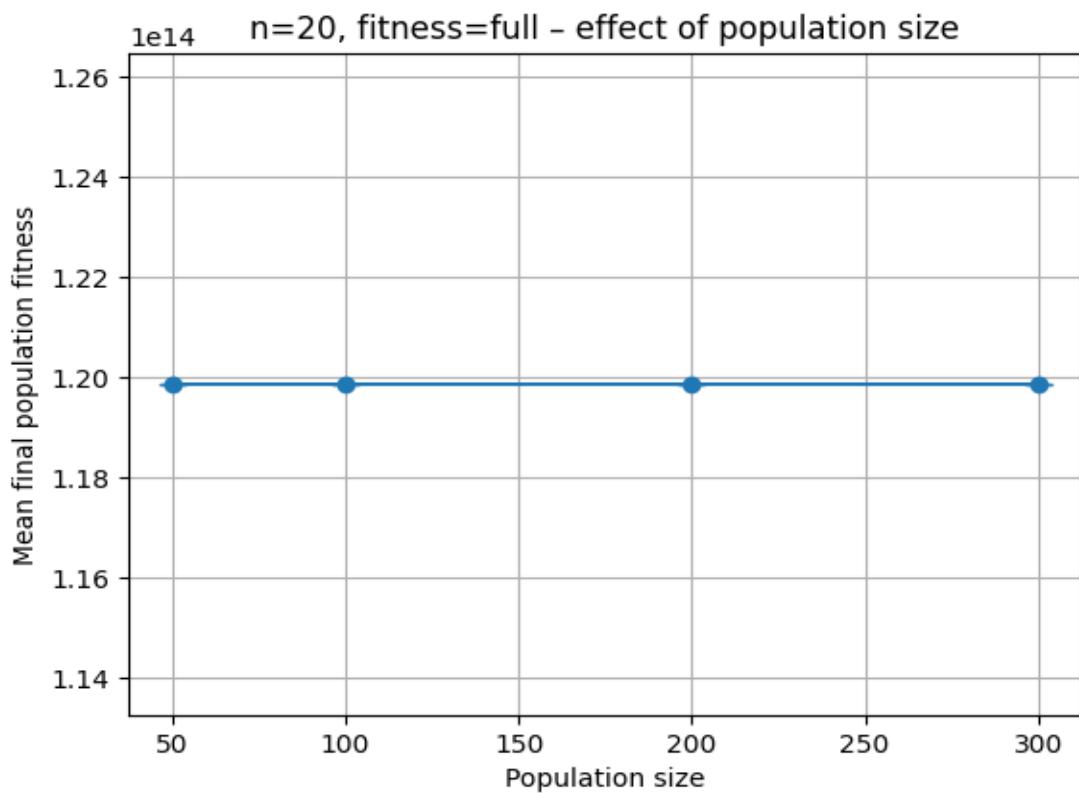
(ث)

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

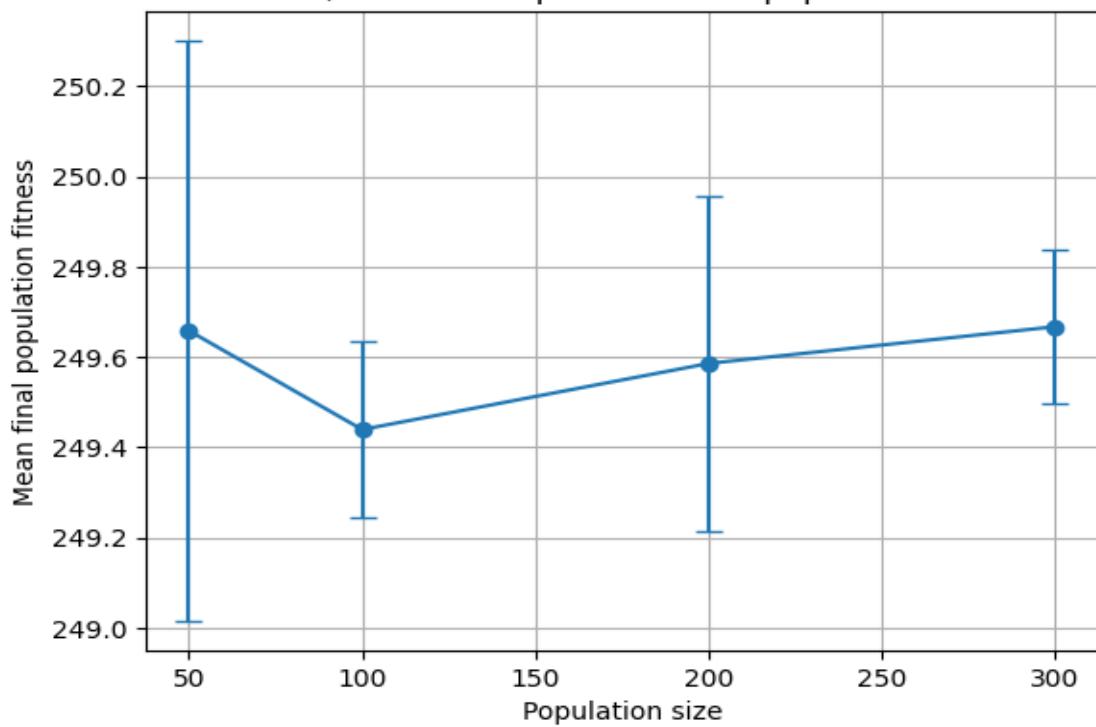
- جمعیت بزرگ‌تر یعنی تنوع بیشتر در ابتدای کار یعنی شанс بیشتری برای پیدا کردن آیتم‌های خوب و فرار از مینیمم‌های محلی؛
- اما هزینه‌ی محاسباتی خطی با اندازه‌ی جمعیت زیاد می‌شود.



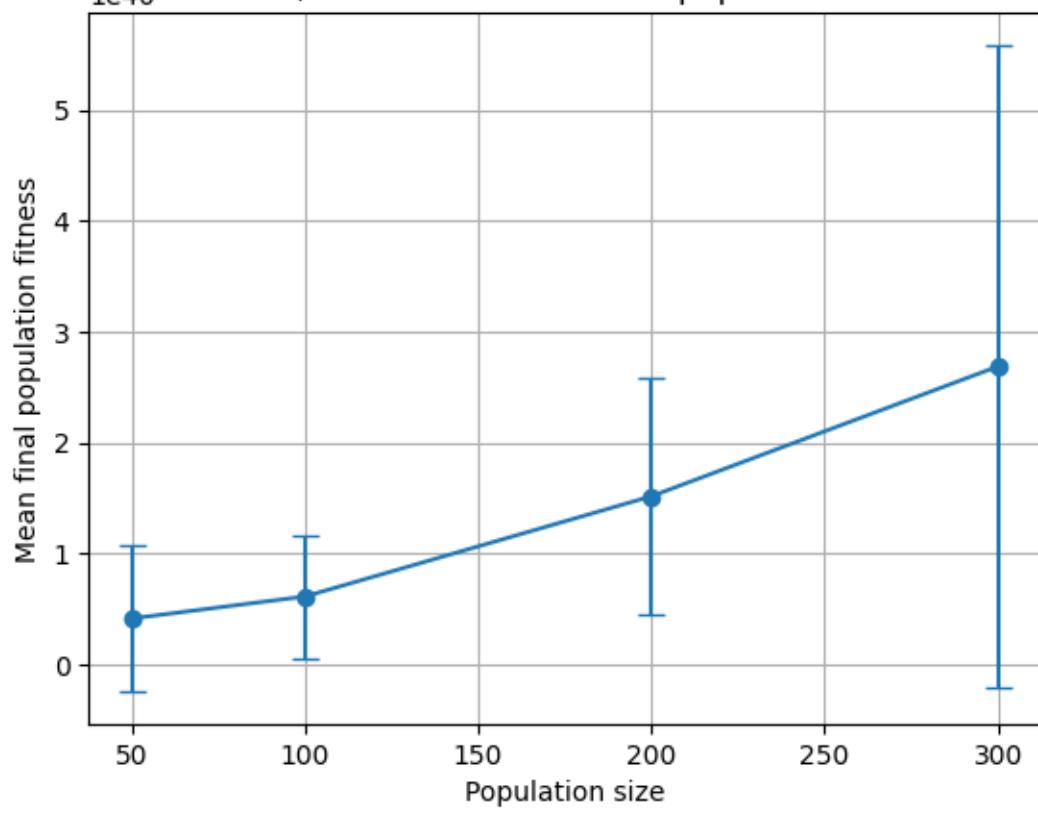


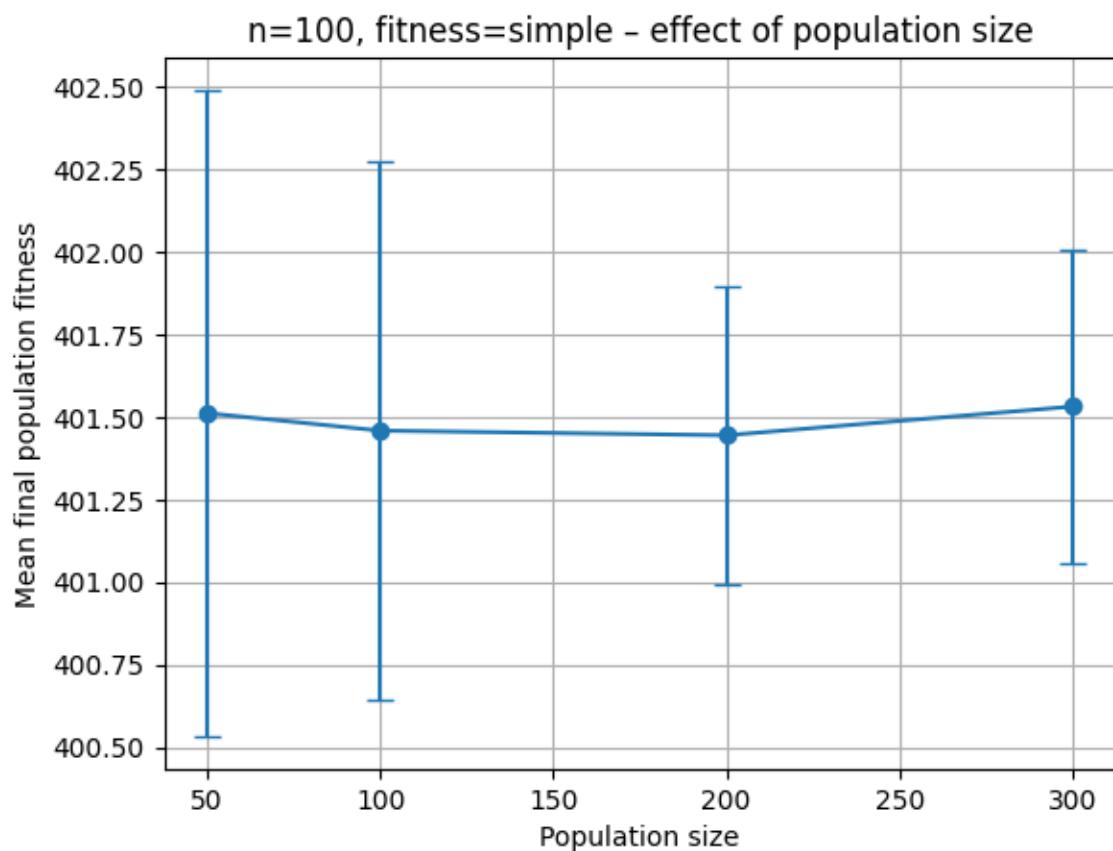
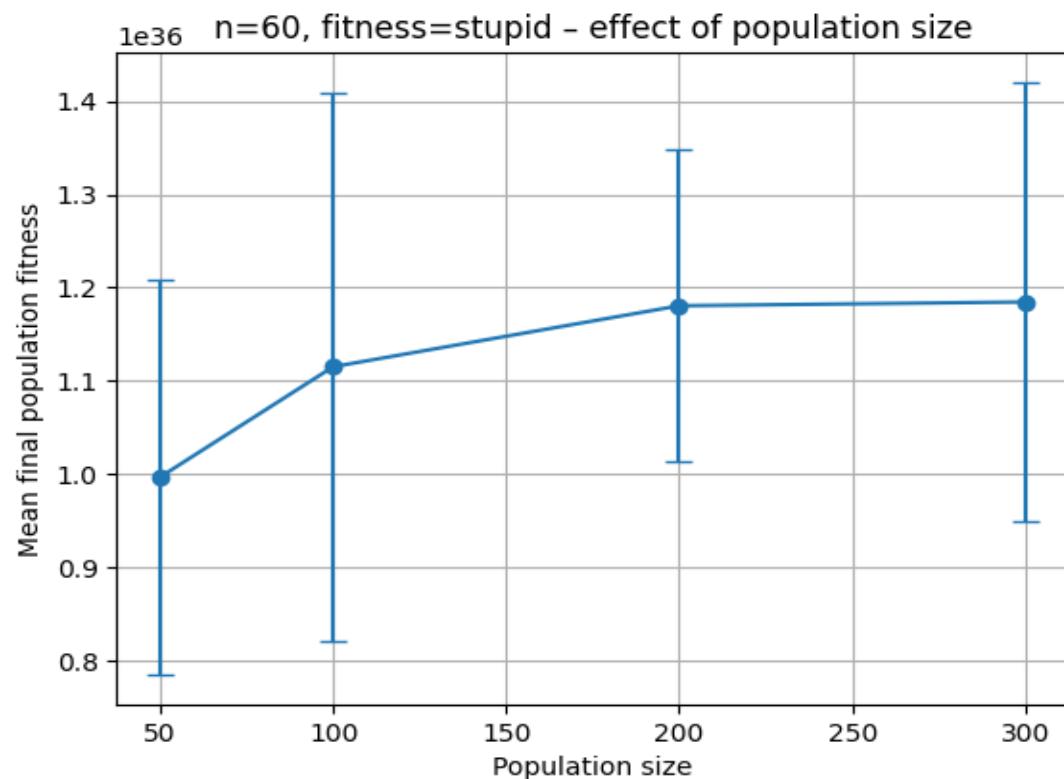


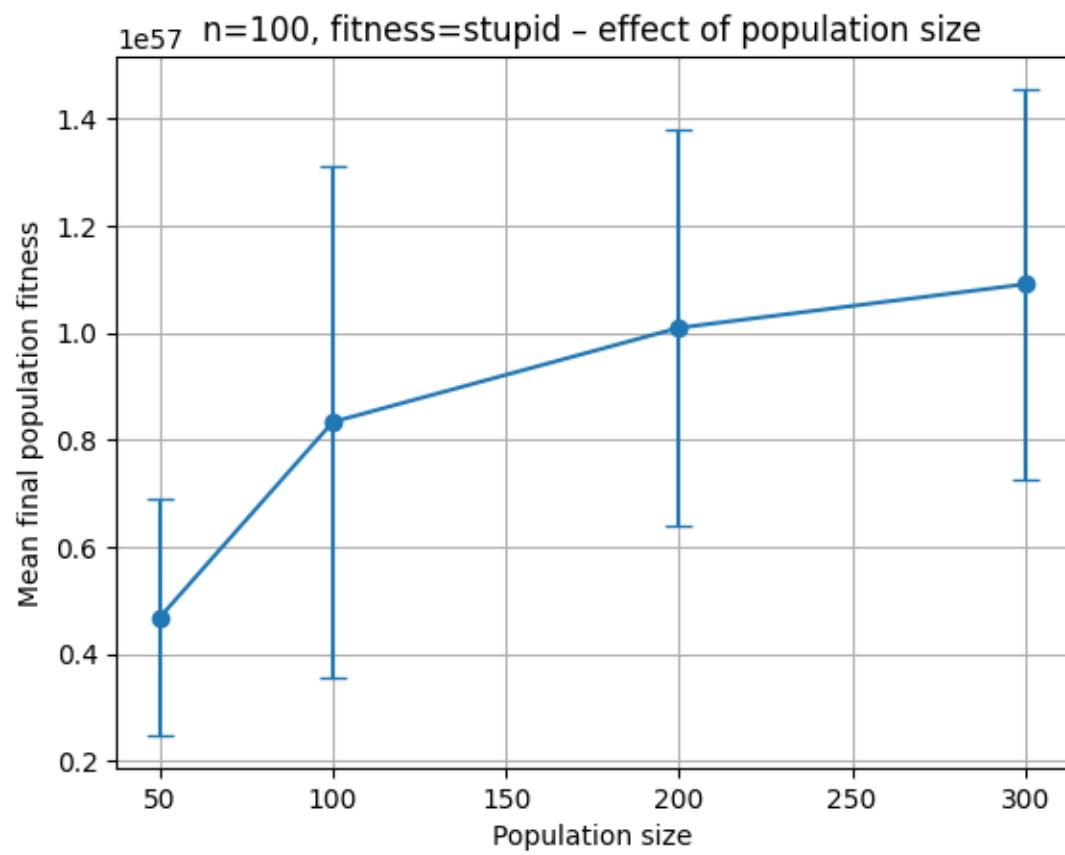
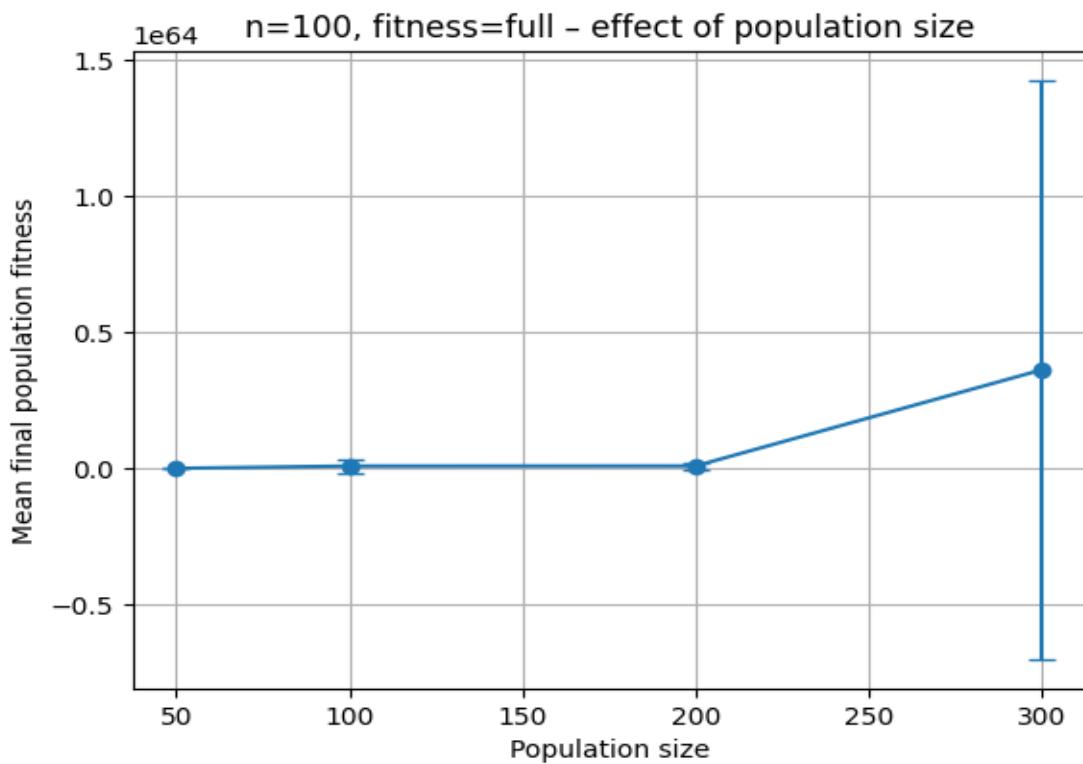
$n=60$, fitness=simple - effect of population size



$n=60$, fitness=full - effect of population size



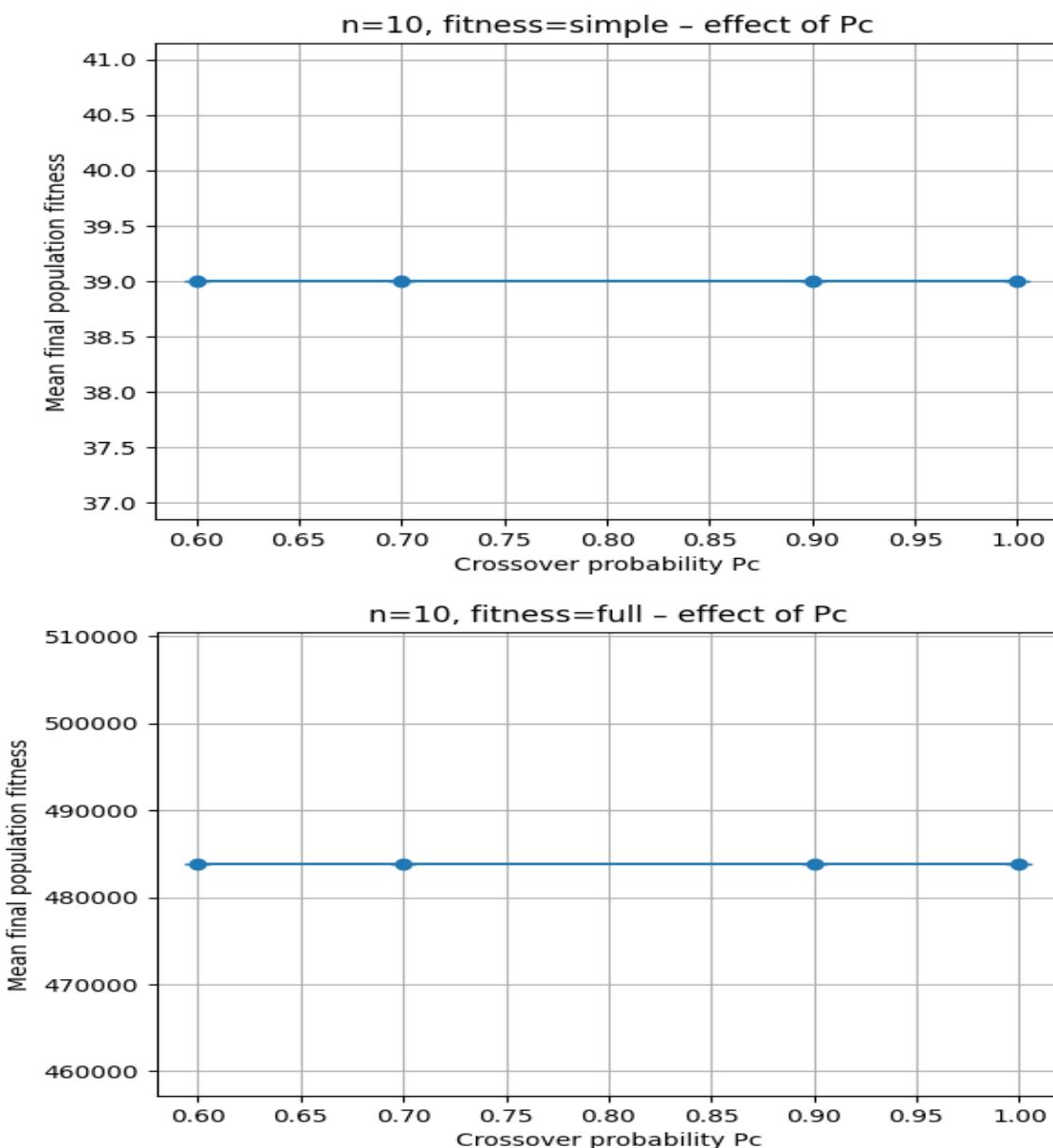


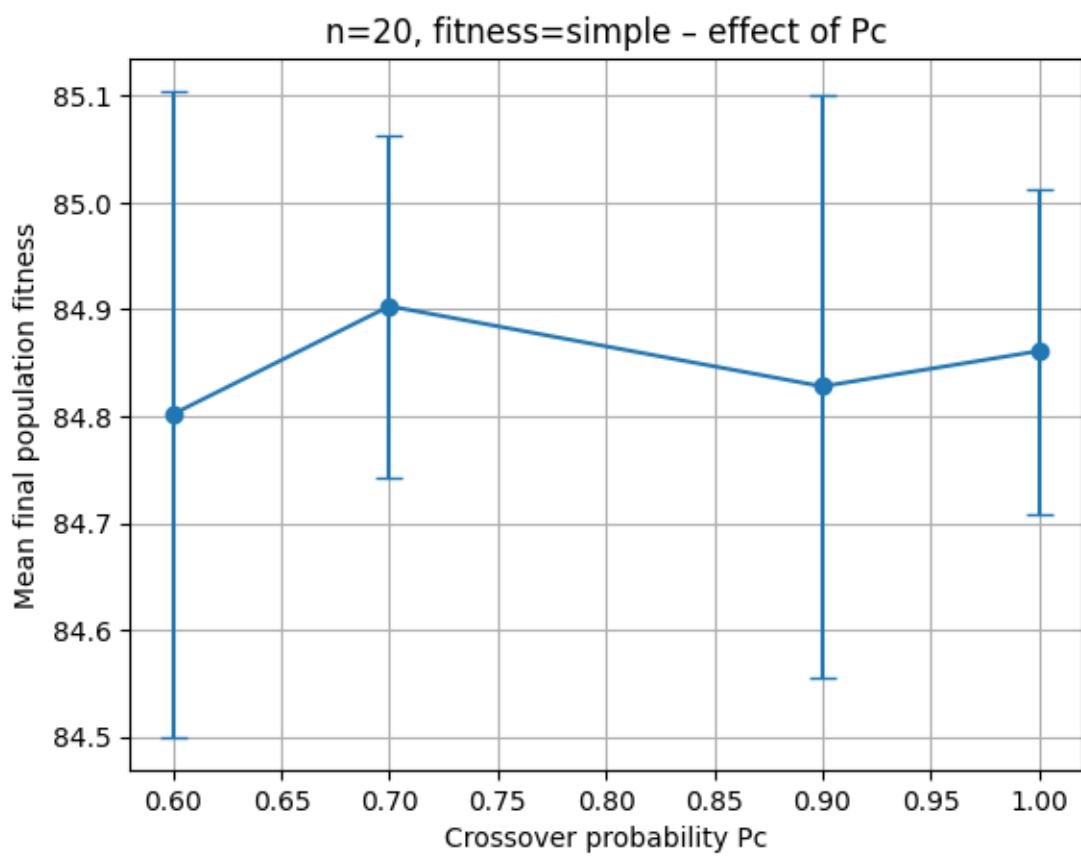
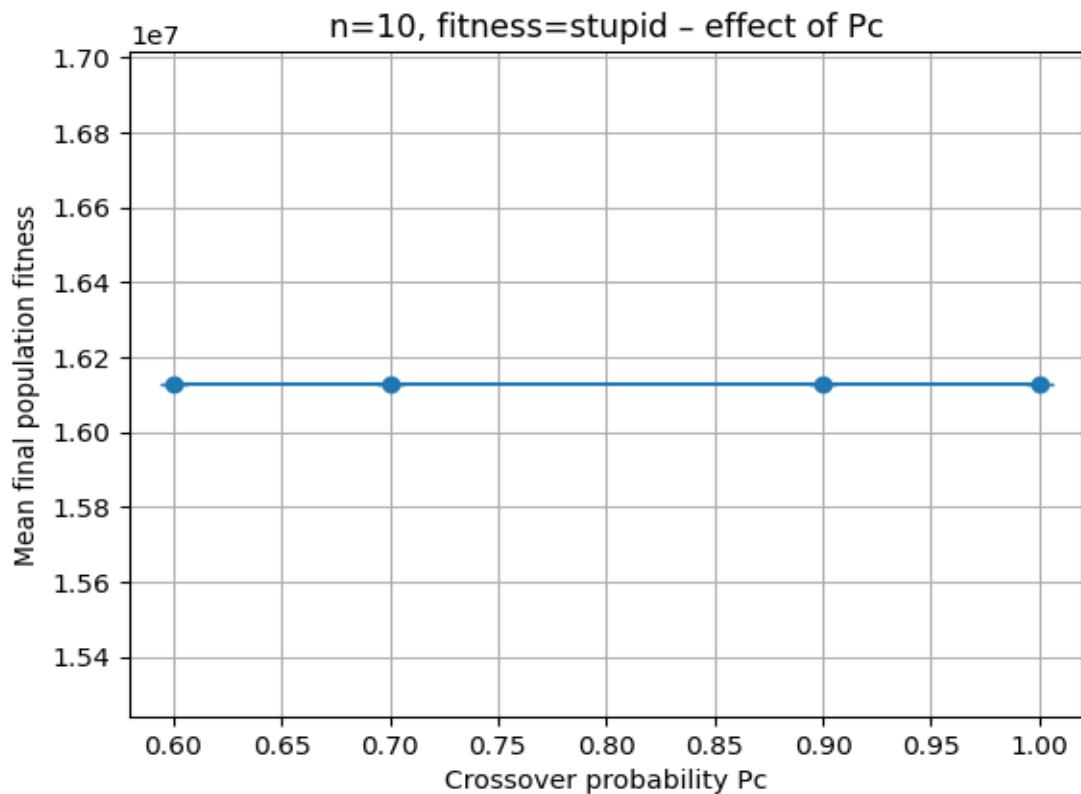


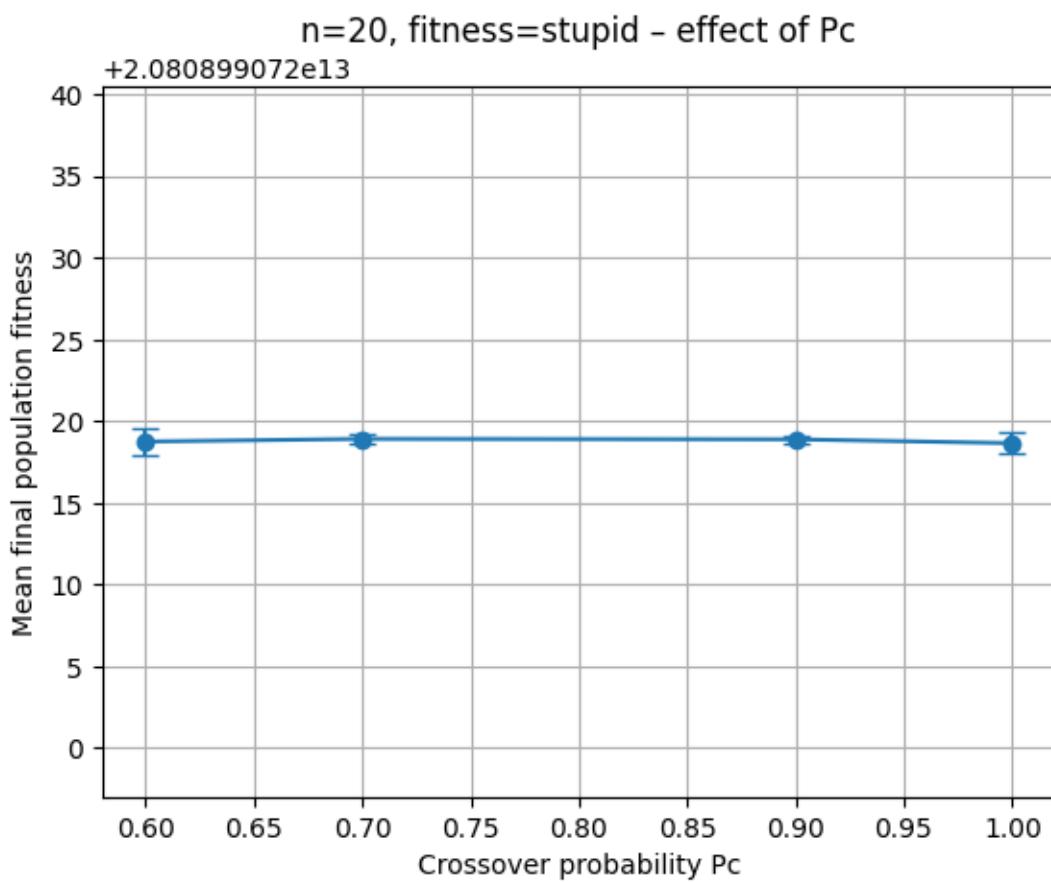
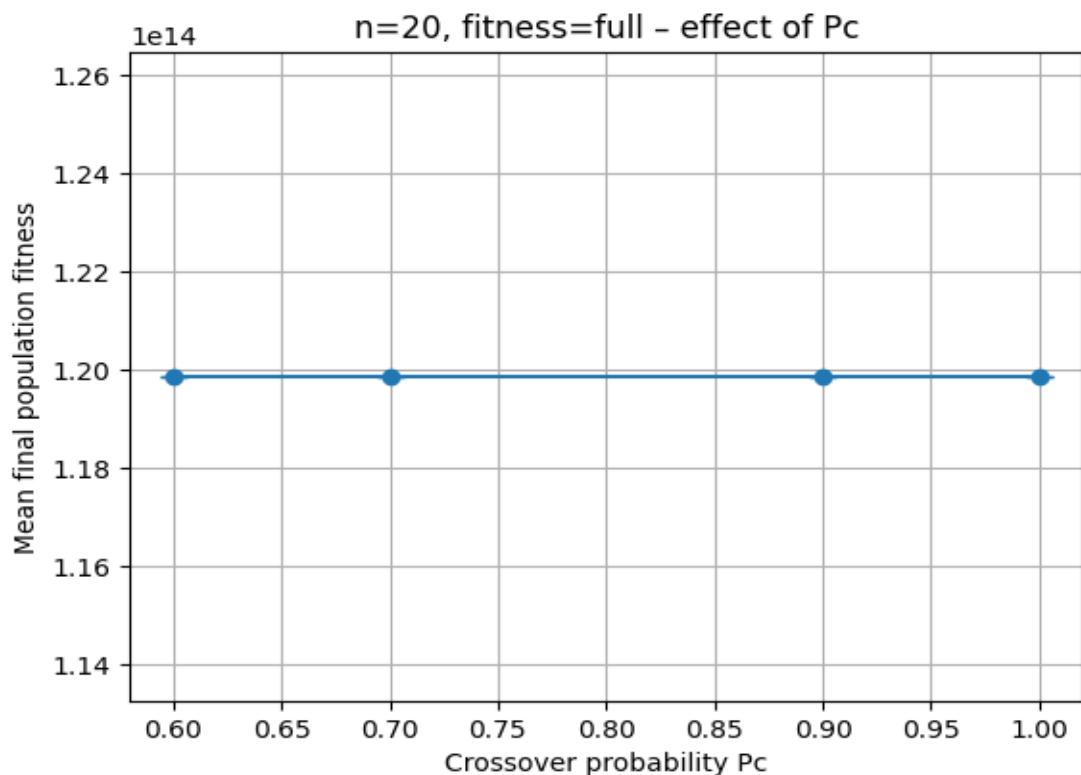
(ج)

احتمال بازترکیب P_C :

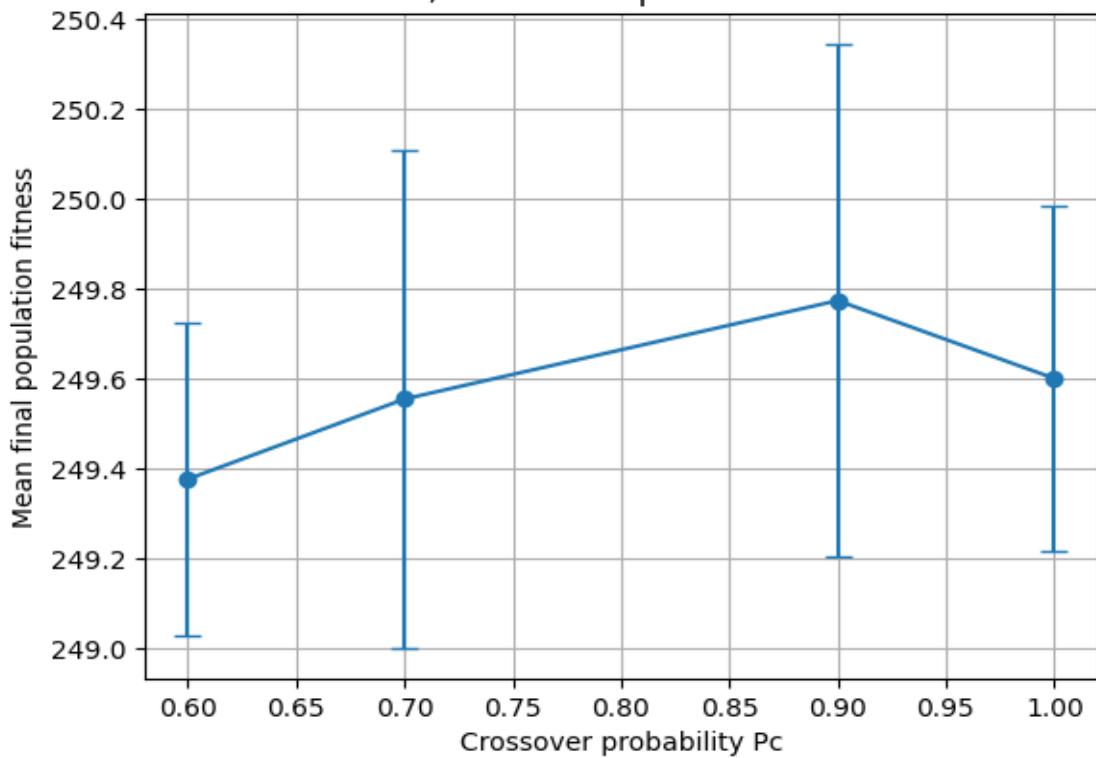
- مقادیر خیلی پایین: استفاده‌ی ناکافی از ترکیب اطلاعات والدین؛ الگوریتم بیشتر شبیه جستجوی تصادفی + جهش است؛
- مقادیر خیلی بالا (نزدیک ۱): همگرایی سریع‌تر ولی در بعضی موارد افزایش خطر همشکلی (loss of diversity).



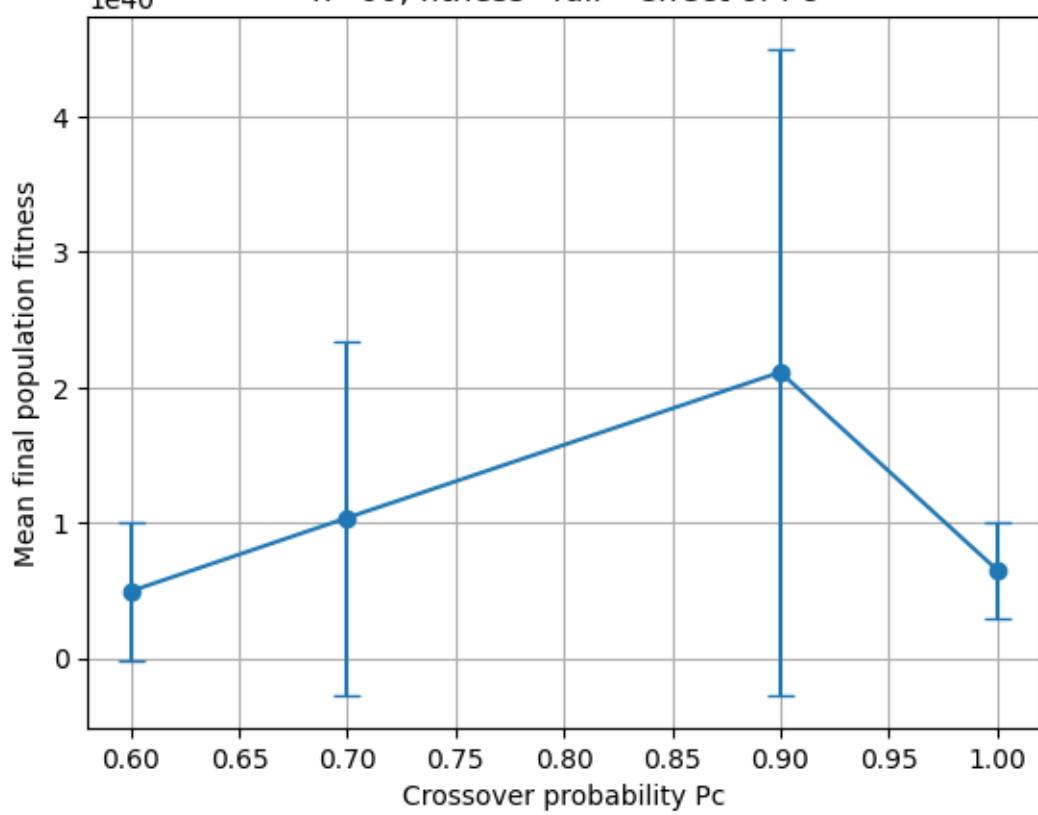




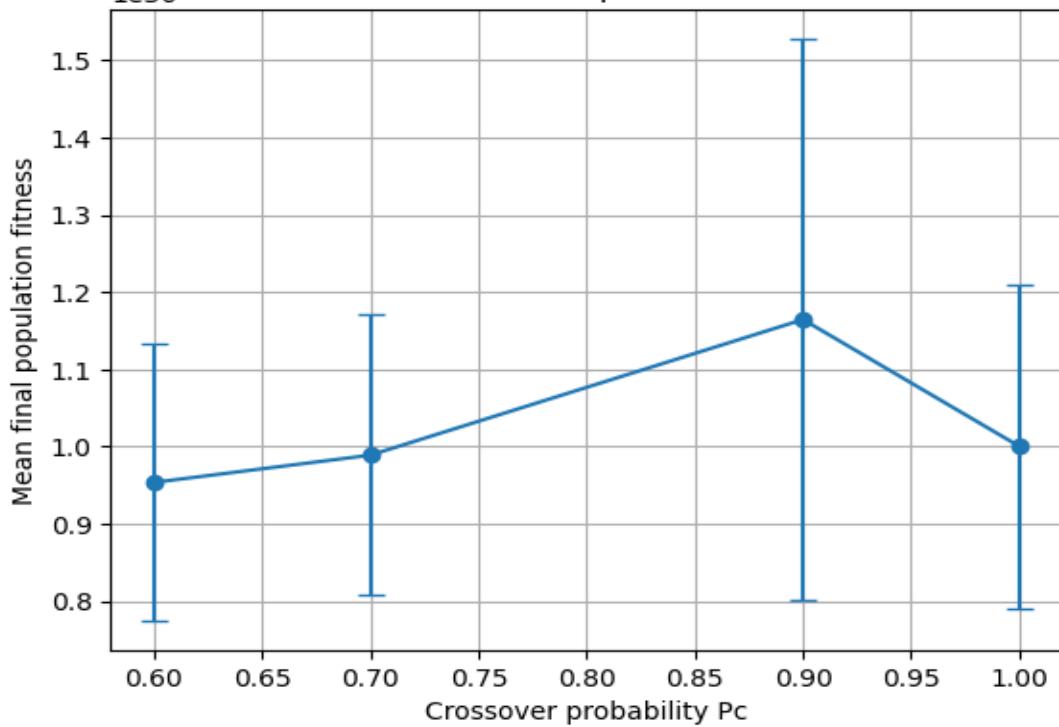
$n=60$, fitness=simple - effect of P_c



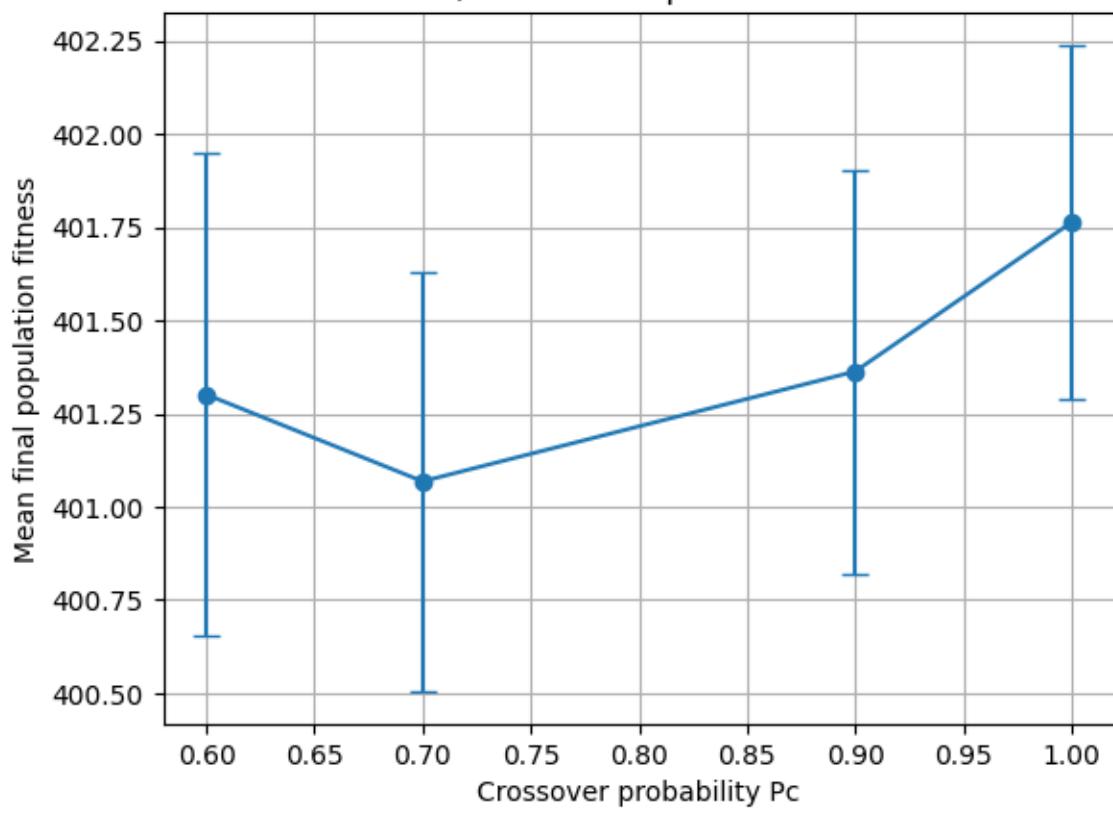
$n=60$, fitness=full - effect of P_c

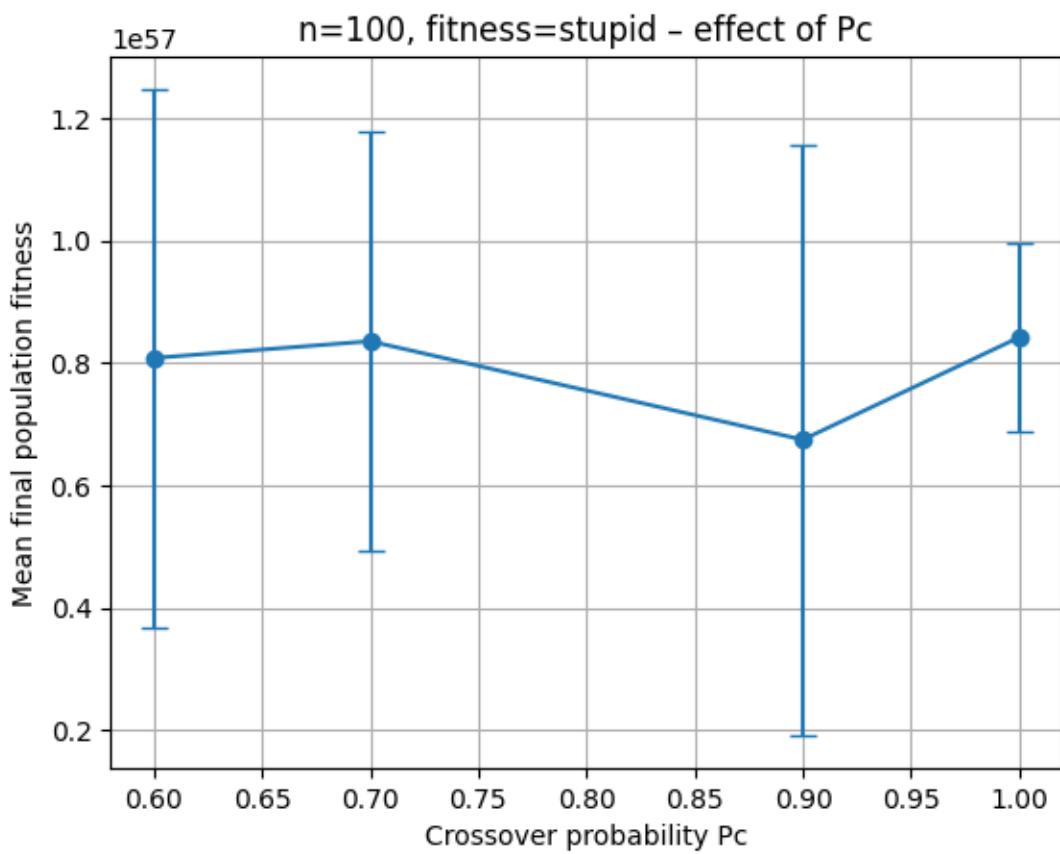
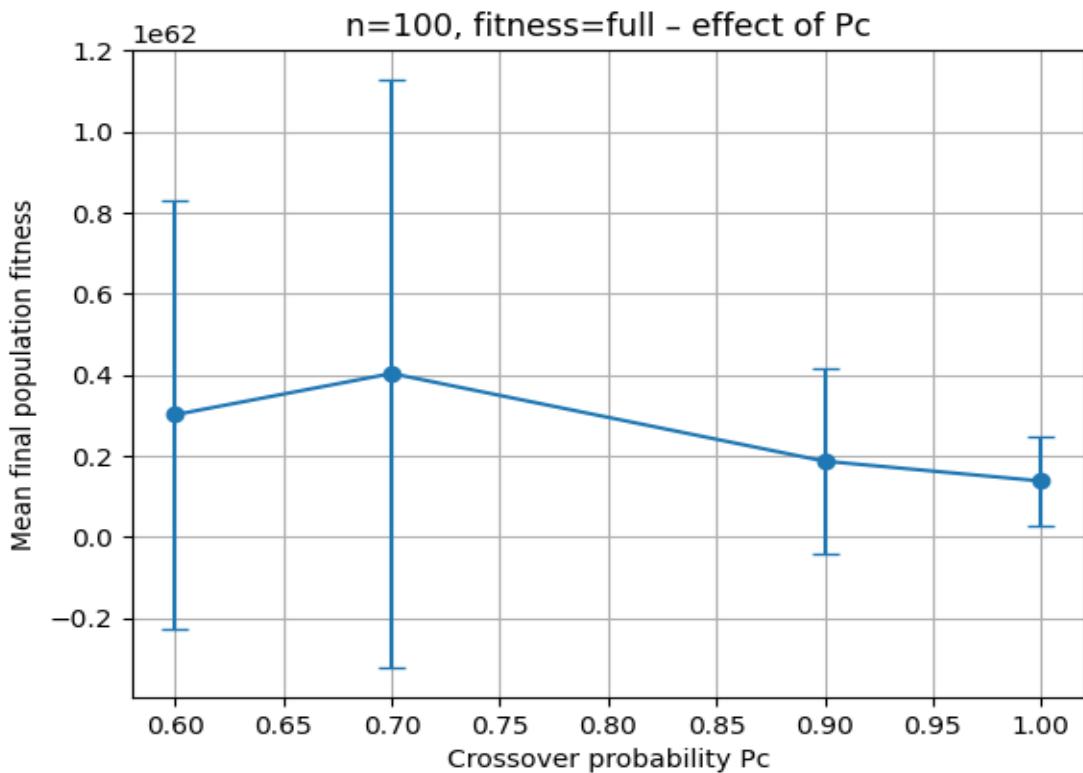


1e36 $n=60$, fitness=stupid - effect of P_c



$n=100$, fitness=simple - effect of P_c

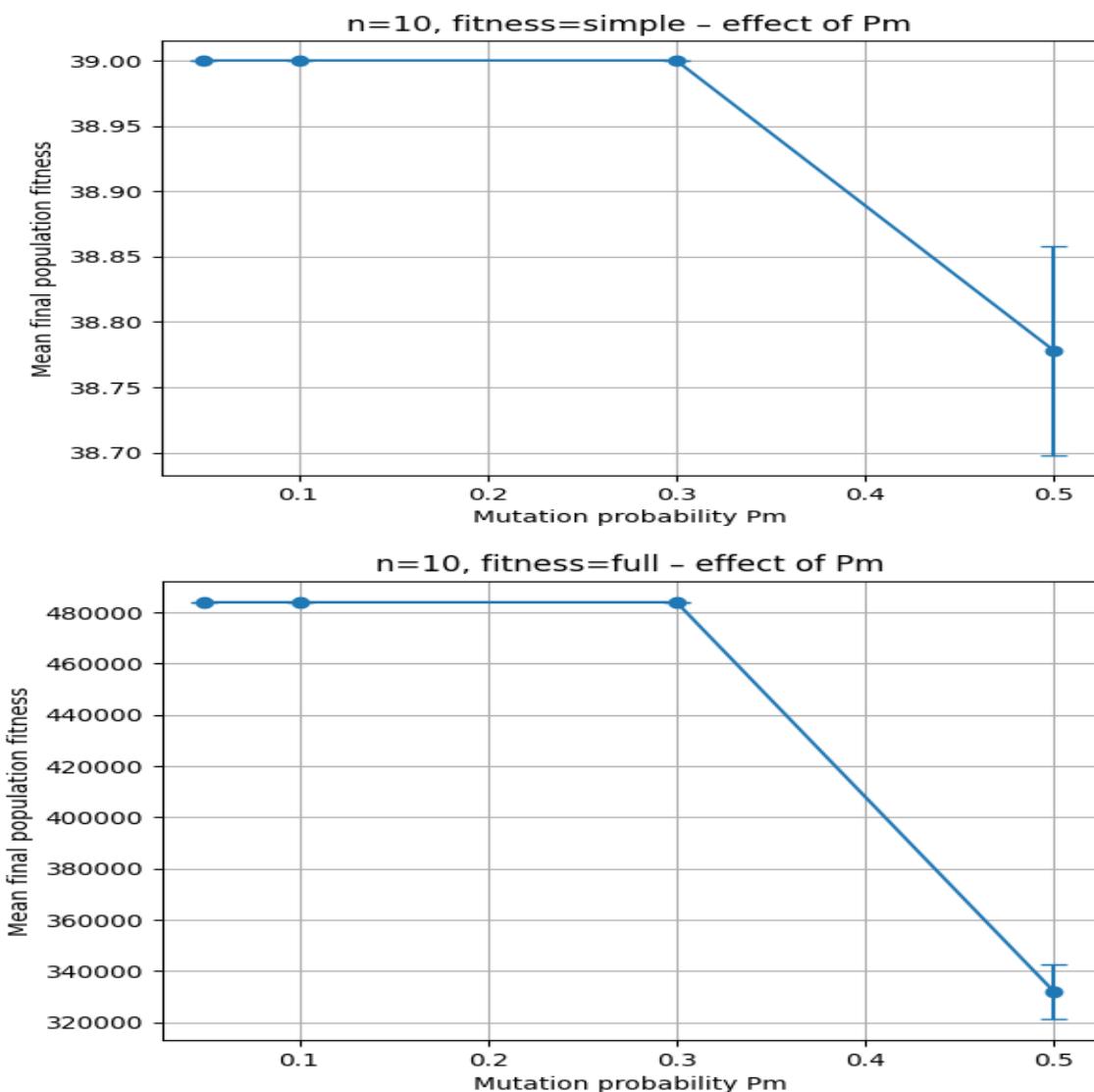


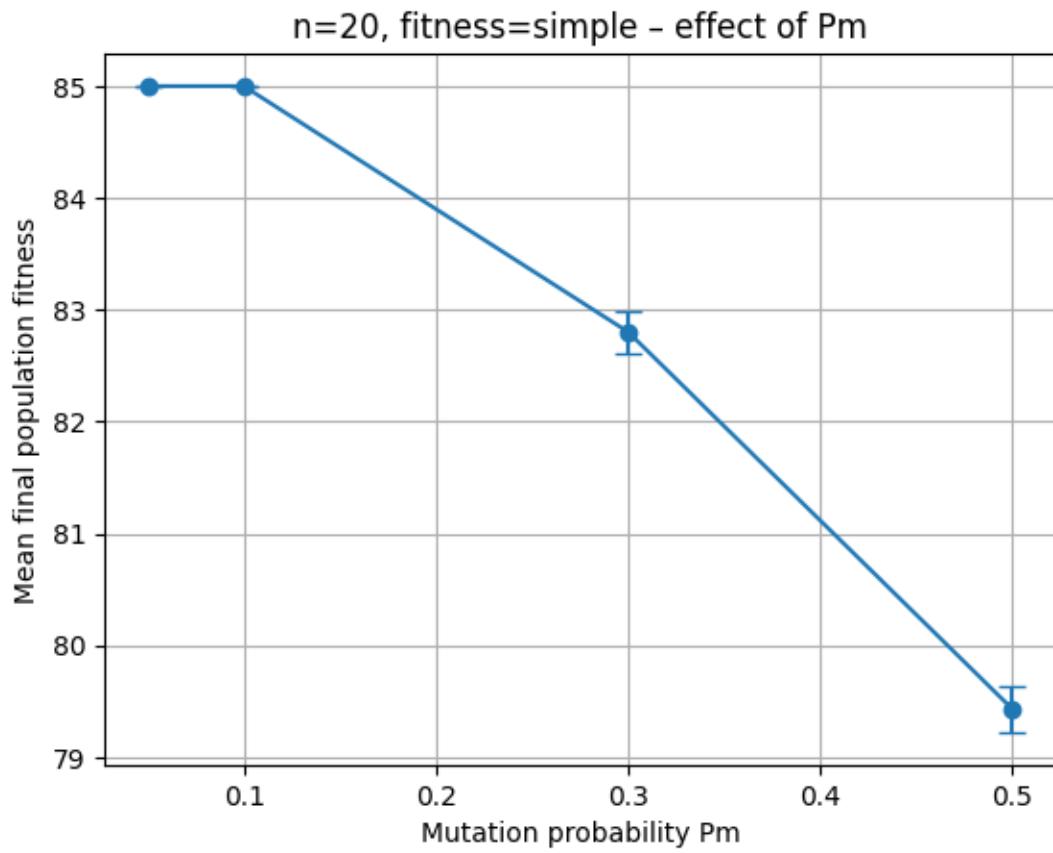
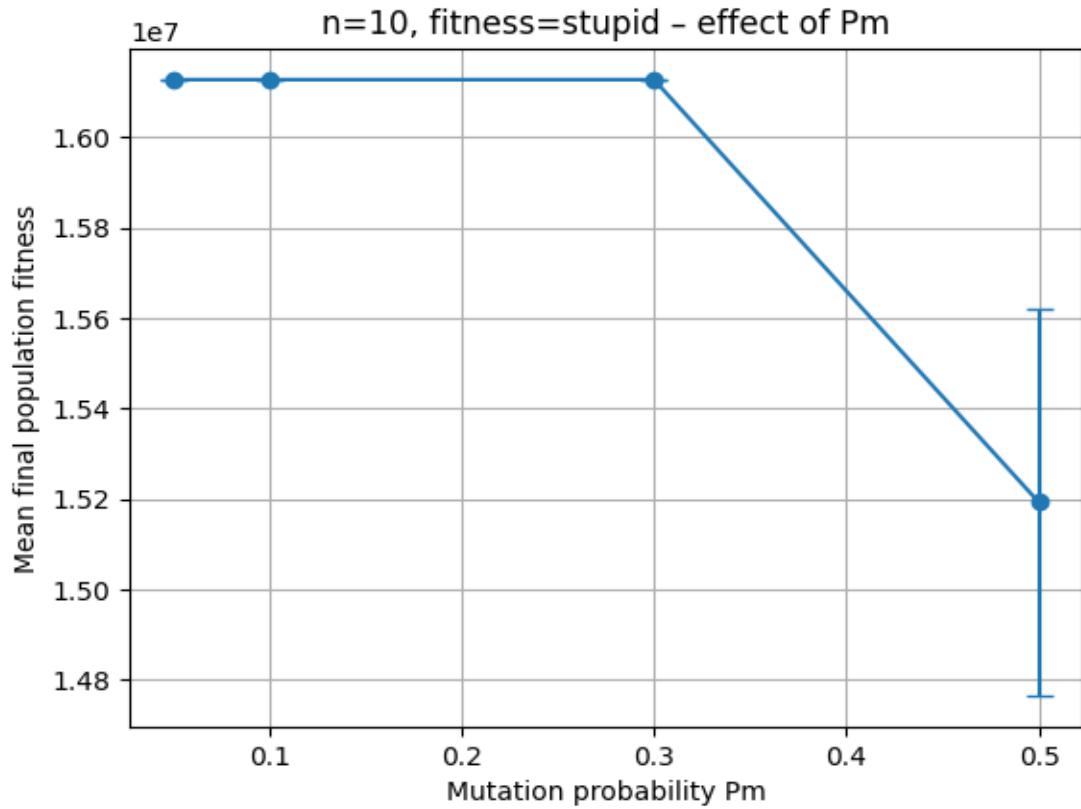


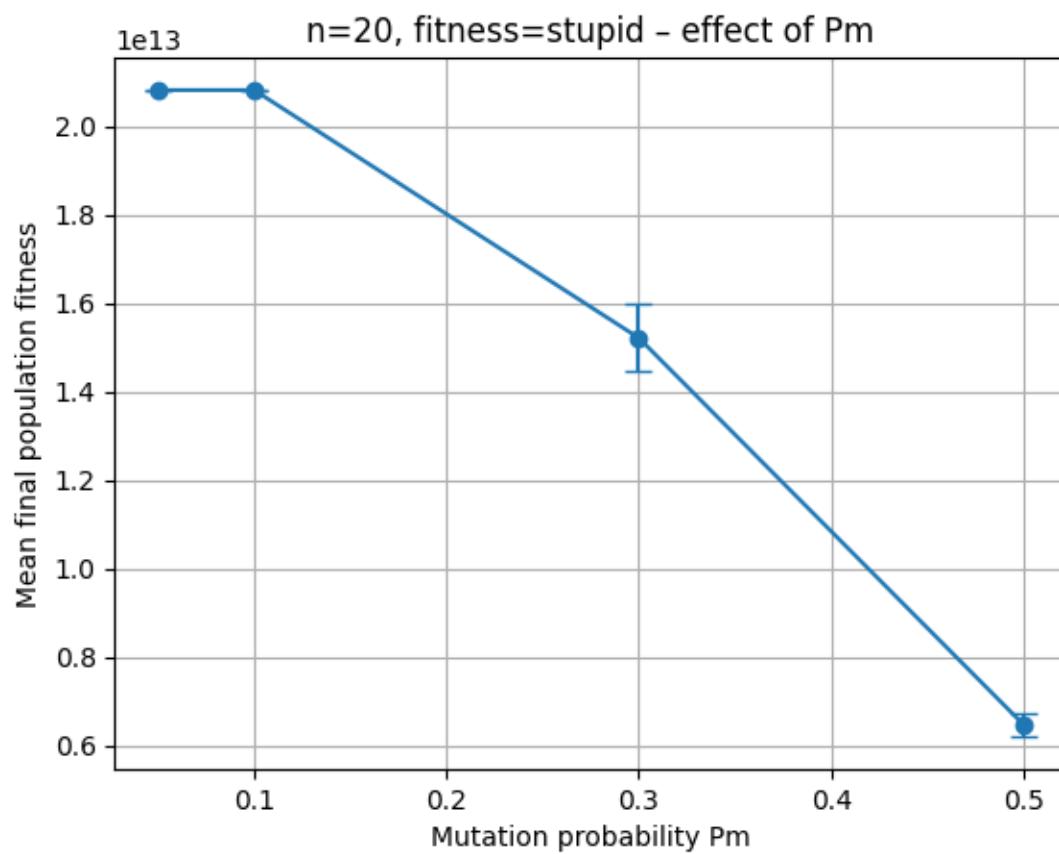
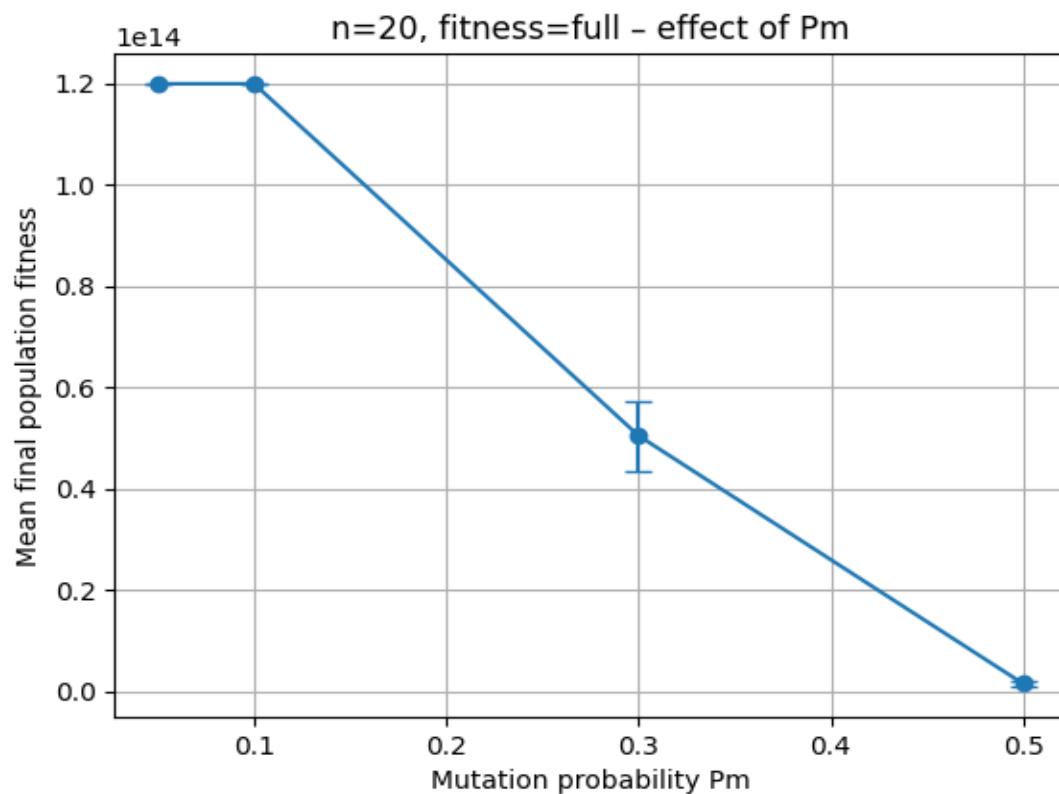
(ج)

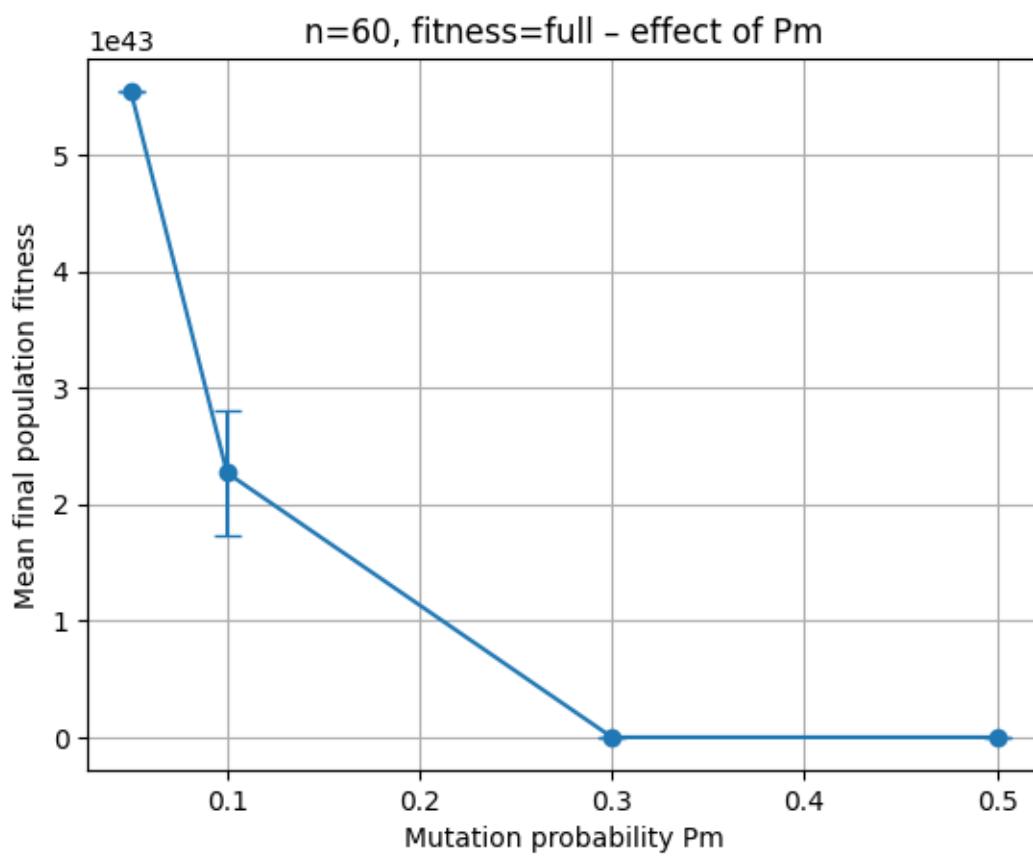
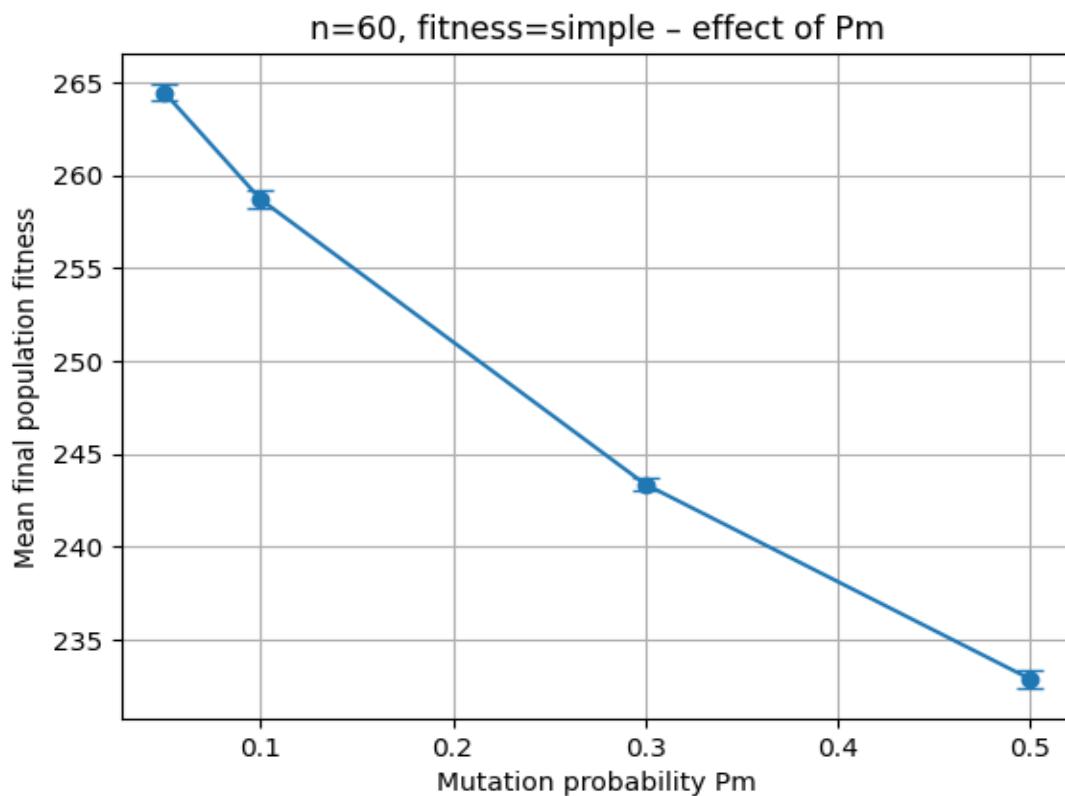
احتمال جهش : P_m

- 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.

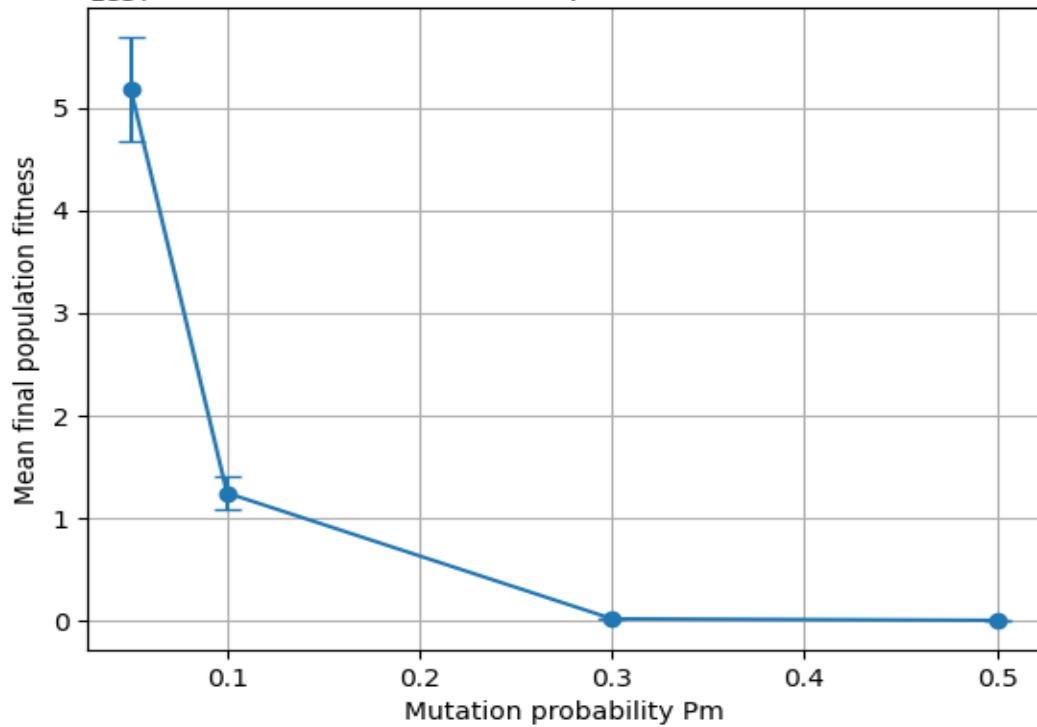




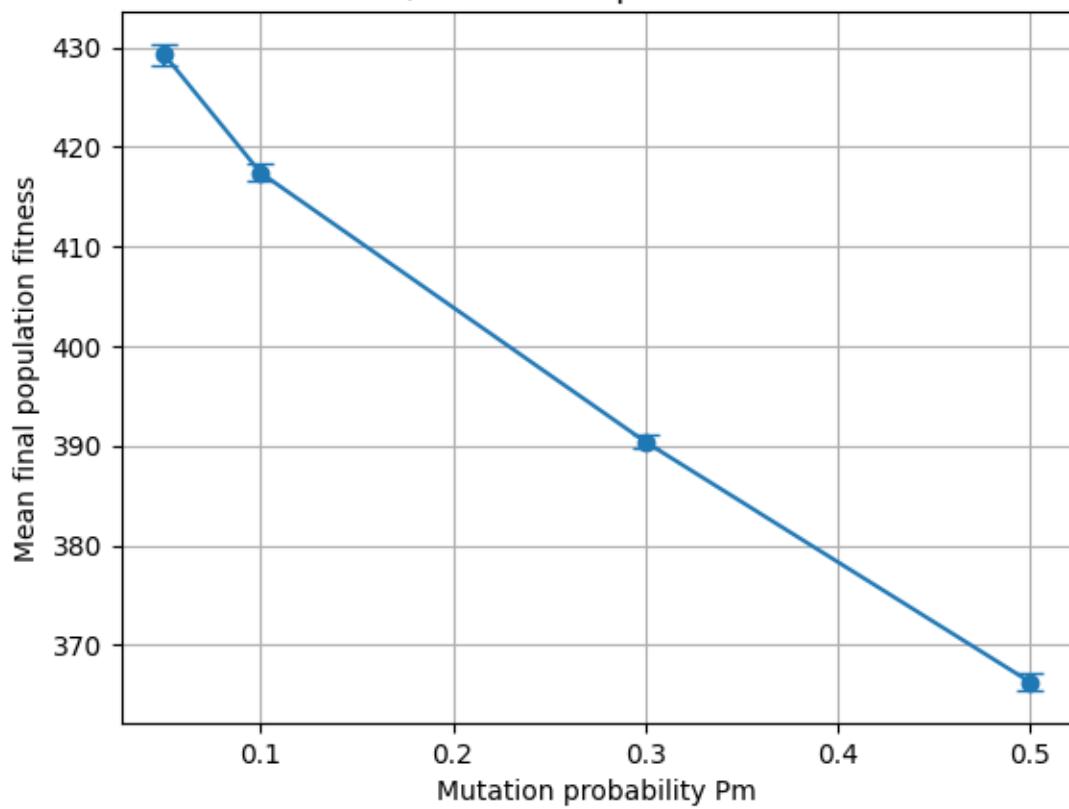




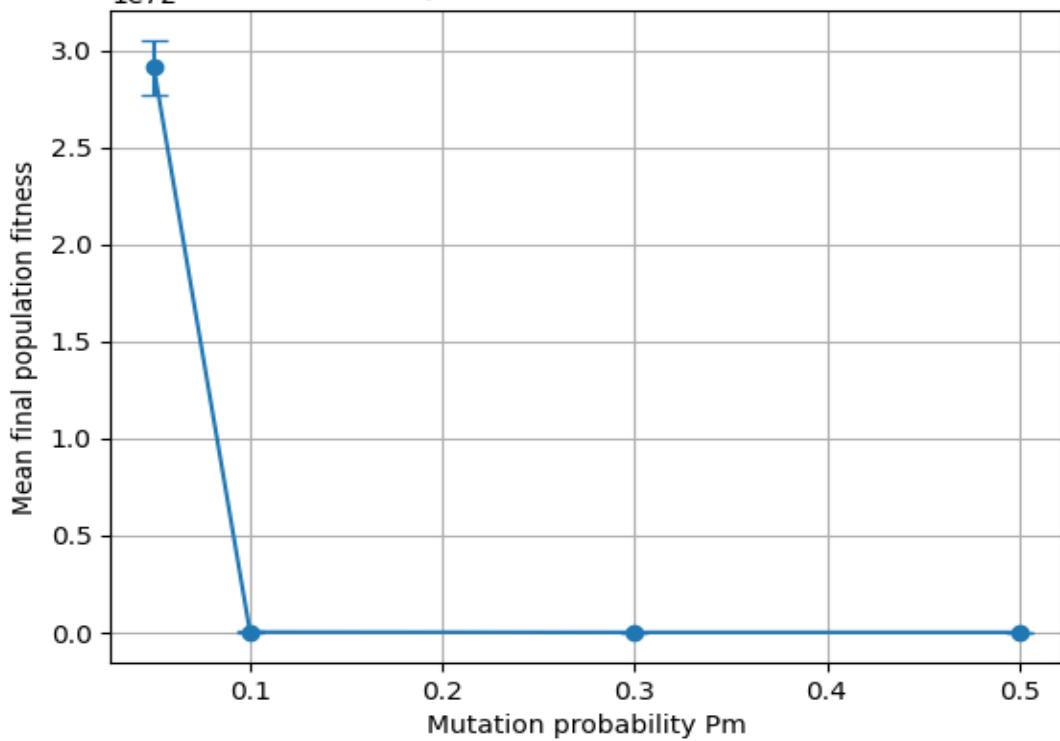
1e37 n=60, fitness=stupid - effect of Pm



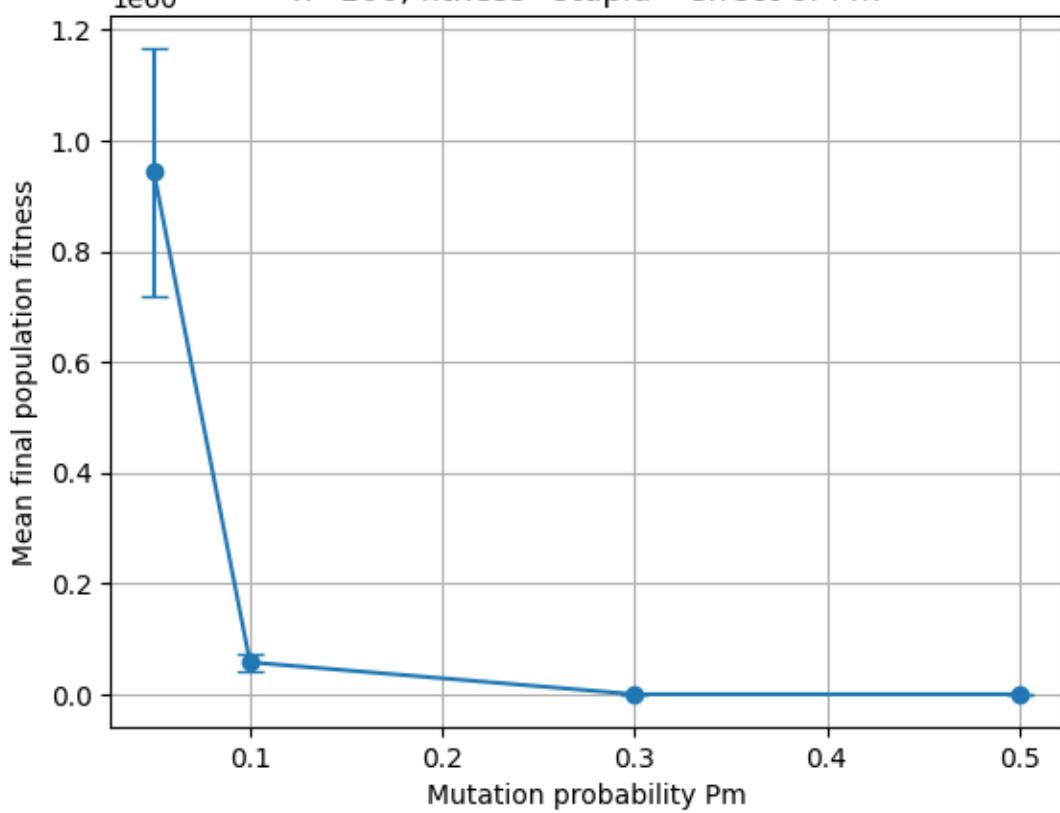
n=100, fitness=simple - effect of Pm



1e72 n=100, fitness=full - effect of Pm



1e60 n=100, fitness=stupid - effect of Pm



(ج)

بهترین راه حل ها:

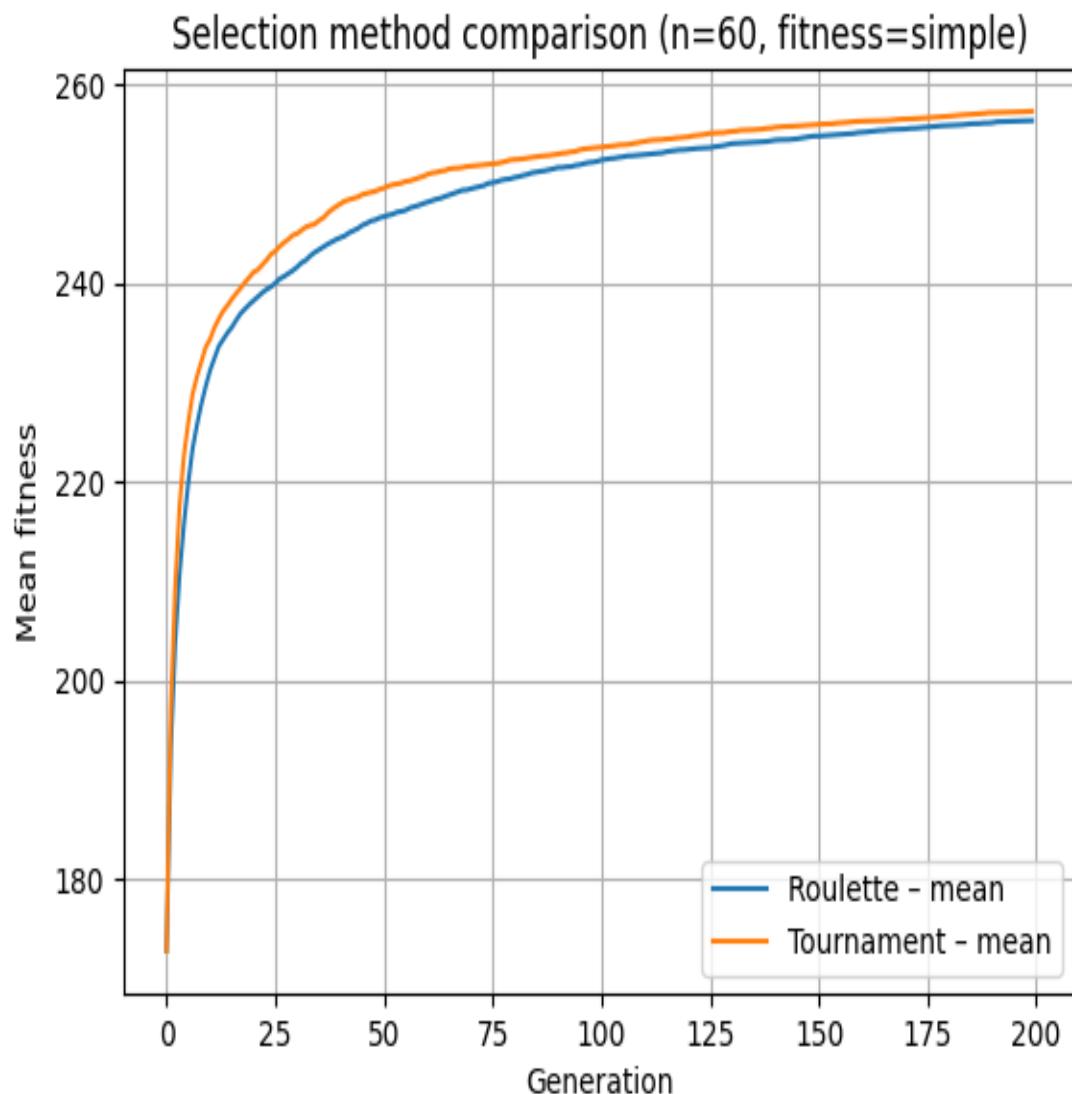
برای هر ترکیب پارامتر و هرتابع، با بررسی final_fitnesses و final_population بیشترین برازنده استخراج شده و کروموزوم متناظر در فایل CSV گزارش شده است (وزن، ارزش، نوع تابع).

question	n	fitness_type	pop_size	max_gen	pc	Pm	best_run	chromos	total_weight	total_value	best_fitness
base_n10	10	simple	100	400	0.8	0.2	0	1.11E+09	22	39	39
base_n10	10	full	100	400	0.8	0.2	0	1.11E+09	24	42	483840
base_n10	10	stupid	100	400	0.8	0.2	0	1.11E+09	22	39	16127961
p_probler	10	simple	100	400	0.8	0.2	0	1.11E+09	22	39	39
p_probler	10	full	100	400	0.8	0.2	0	1.11E+09	24	42	483840
p_probler	10	stupid	100	400	0.8	0.2	0	1.11E+09	22	39	16127961
p_probler	20	simple	100	400	0.8	0.2	0	1101111111	39	85	85
p_probler	20	full	100	400	0.8	0.2	0	1111111111	57	109	1.2E+14
p_probler	20	stupid	100	400	0.8	0.2	0	1111011111	39	81	2.08E+13
p_probler	60	simple	100	400	0.8	0.2	4	1111111111	129	256	256
p_probler	60	full	100	400	0.8	0.2	0	1011111111	181	325	5.28E+41
p_probler	60	stupid	100	400	0.8	0.2	2	0111010111	128	258	7.22E+36
p_probler	100	simple	100	400	0.8	0.2	2	1111101111	213	414	414
p_probler	100	full	100	400	0.8	0.2	2	1111111111	272	497	4.66E+64
p_probler	100	stupid	100	400	0.8	0.2	1	0011111111	213	413	9.90E+57
t_generat	10	simple	100	50	0.8	0.2	0	1.11E+09	22	39	39
t_generat	10	full	100	50	0.8	0.2	0	1.11E+09	24	42	483840
t_generat	10	stupid	100	50	0.8	0.2	0	1.11E+09	22	39	16127961
t_generat	10	simple	100	100	0.8	0.2	0	1.11E+09	22	39	39
t_generat	10	full	100	100	0.8	0.2	0	1.11E+09	24	42	483840
t_generat	10	stupid	100	100	0.8	0.2	0	1.11E+09	22	39	16127961
t_generat	10	simple	100	200	0.8	0.2	0	1.11E+09	22	39	39

(خ)

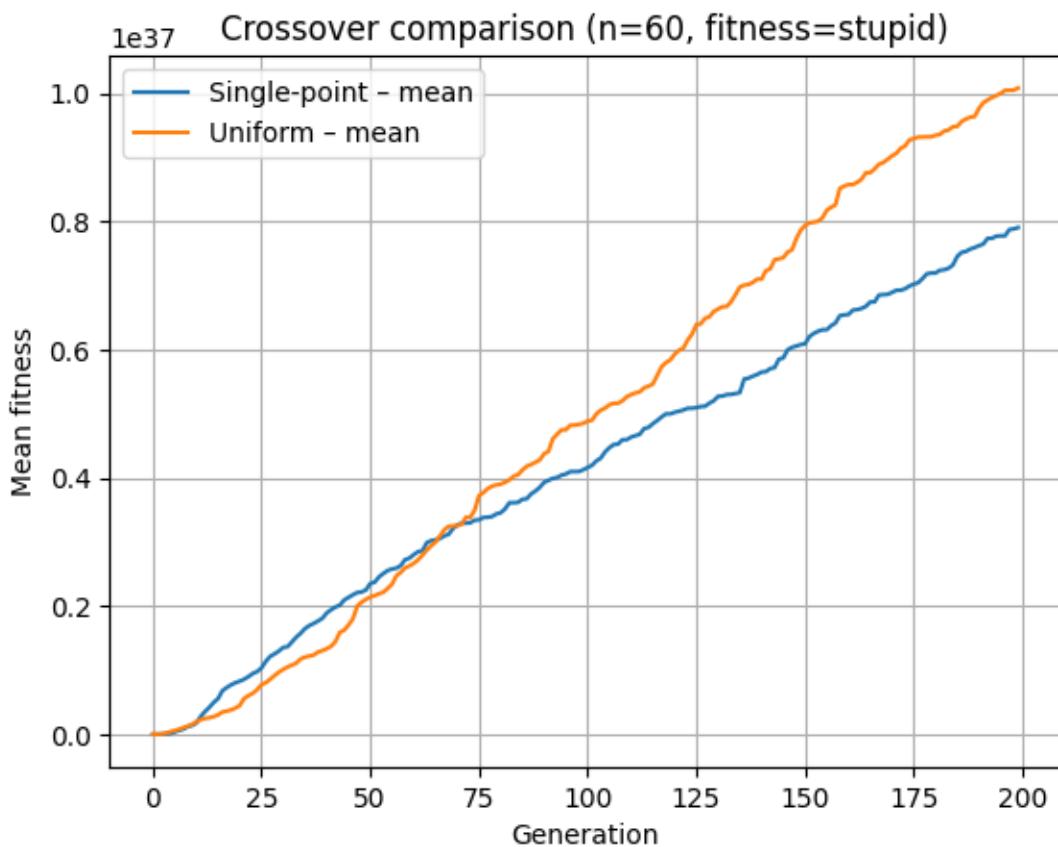
: Tournament Roulette با Tournament نشان داد که مقایسه‌ی انتخاب

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



(د)

در بازترکیب یکنواخت، برای هر ژن به طور مستقل با احتمال ۰.۵ از والد ۱ یا ۲ انتخاب می‌شود. در نمودارها می‌بینیم که Uniform نوع 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 راه حل»، کروموزوم دودویی‌ای است که بیشترین برازنده‌گی را در پایان هر اجرا داشته است (برای هر تابع برازنده‌گی، هر n و هر تنظیم پارامتر).

- این کار را برای:

- سه تابع برازنده‌گی (simple, full, stupid)
 - چهار اندازه‌ی مسئله ۱۰، ۲۰، ۶۰، ۱۰۰
 - و تنظیمات مختلفی که در بخش‌های پ، ت، ث، ج، چ امتحان کردیم،
 - تکرار کرده و بردار دودویی بهترین فرد و مقدار برازنده‌گی اش 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 برازنده‌گی نهایی بالاتری رسیده است.
- از نظر تئوری:
- Tournament selection ○
 - Roulette دارد.
 - همگرایی سریع‌تر (بهترین‌ها زودتر غالب می‌شوند)
 - ولی احتمال همگرایی نابهنجام و کاهش شدید تنوع جمعیت هم بیشتر است.
 - روی نمودارها می‌بینیم که میانگین برازنده‌گی در روش Tournament سریع‌تر بالا می‌رود، اما در نسل‌های بعد ممکن است روی مقدار متوسطتری قفل شود، در حالی که Roulette کندر ولی نرم‌تر ولی 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، تمايل 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 جستجو 2^n به صورت نمایی بزرگ می‌شود. EA برای n ‌های بزرگ‌تر معمولاً به جواب‌های خیلی خوب می‌رسد ولی نیاز به جمعیت بزرگ‌تر و نسل‌های بیشتر دارد.

میانگین برازنده‌گی 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 ۵۰) رشد خیلی سریع ولی ناقص است؛ الگوریتم هنوز به تعادل نرسیده است. بین ۱۰۰ تا ۲۰۰ نسل معمولاً بخش عمده‌ی بهبود انجام می‌شود؛ بعد از آن نمودار بهترین/میانگین برازنده‌گی تقریباً صاف می‌شود (باذده نزولی).

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

- اندازه جمعیت (popSize) :

جمعیت کوچک یعنی تنوع کم، فشار راندمان بالا ولی احتمال گیر کردن در بهینه های محلی زیاد. جمعیت خیلی بزرگ یعنی تنوع زیاد، ولی هزینه های محاسباتی بالا و سرعت همگرایی پایین تر.

معمولًاً یک مقدار میانی (مثلاً ۲۰۰-۱۰۰) بهترین trade-off را می دهد.

- احتمال بازترکیب P_c :

خیلی پایین یعنی بازترکیب کم، افراد تقریباً شبیه والدین می مانند و جستجو بیشتر به جهش وابسته می شود.

خیلی بالا (نزدیک ۱) یعنی تقریباً همهی والدین بازترکیب می شوند؛ اگر جمعیت خوب باشد، این کمک می کند ترکیب های قوی تر ساخته شود؛ اگر جمعیت ضعیف باشد، ساختارهای بد هم مرتب مخلوط می شوند.

معمولًاً مقادیر بین ۰.۹ تا ۰.۸ خوب عمل می کنند.

- احتمال جهش P_m :

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

خیلی زیاد (مثلاً ۰.۵) یعنی هر نسل افراد تقریباً به طور تصادفی دوباره ساخته می شوند، الگوریتم شبیه random search می شود.

مقدار میانی (مثلاً ۰.۱ یا ۰.۰۵) معمولًاً بهترین است: تنوع کافی بدون نایود 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 شبیه حرکت روی چشم‌انداز برازنده‌گی 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 knapsack ها 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 برازنده‌گی (Simple, Full, Stupid) :

Simple : منظره‌ی برازنده‌گی نسبتاً نرم و خطی‌تر؛ EA به راحتی ترکیب‌های نزدیک به بهینه را پیدا می‌کند.

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

Stupid : به‌خاطر ضرب و ضریب n^2 مقادیر برازنده‌گی 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 (۲۲، ۳۹، ۱۳۰، ۲۱۱) 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 برازنده‌گی شده است.

n	fitness_type	chromosome (binary)	total_weight	total_value	best_fitness
10	simple	0111111111	22	39	39
10	full	1111111111	24	42	4.84×10^5
10	stupid	1111111101	22	39	1.61×10^7
20	simple	1101111111110001101	39	85	85
20	full	11111111111111111111	57	109	1.20×10^{14}
20	stupid	111101110110011111	39	81	2.08×10^{13}
60	simple	1110111110110110101011110 10111110101101000110011111 1111	130	252	252
60	full	11111111111101111101111111 01111111111111111111111111 1011	174	318	7.55×10^{40}
60	stupid	111011011111110100111111 101001101111011011110111111 0001	130	254	2.44×10^{36}
100	simple	110111011110101110110111 001101110011011011101110111 11110101101111101010111111 1110011110110110	211	414	414
100	full	111101110101110111101111 1111101111101111111111111101111 110111111111111111111111010101 1110111111111111	264	478	2.36×10^{62}
100	stupid	11111010111111011111111100 010110101001101111110111010 111111001111111010101111010 1111010111010111	211	411	3.46×10^{57}

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

• روش انتخاب: چرخ‌گردونه (Roulette Wheel)

popSize = 100 •

MaxGenerations = 400 •

Pc = 0.8 •

Pm = 0.2 •

۳- بخش ۲: 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؛

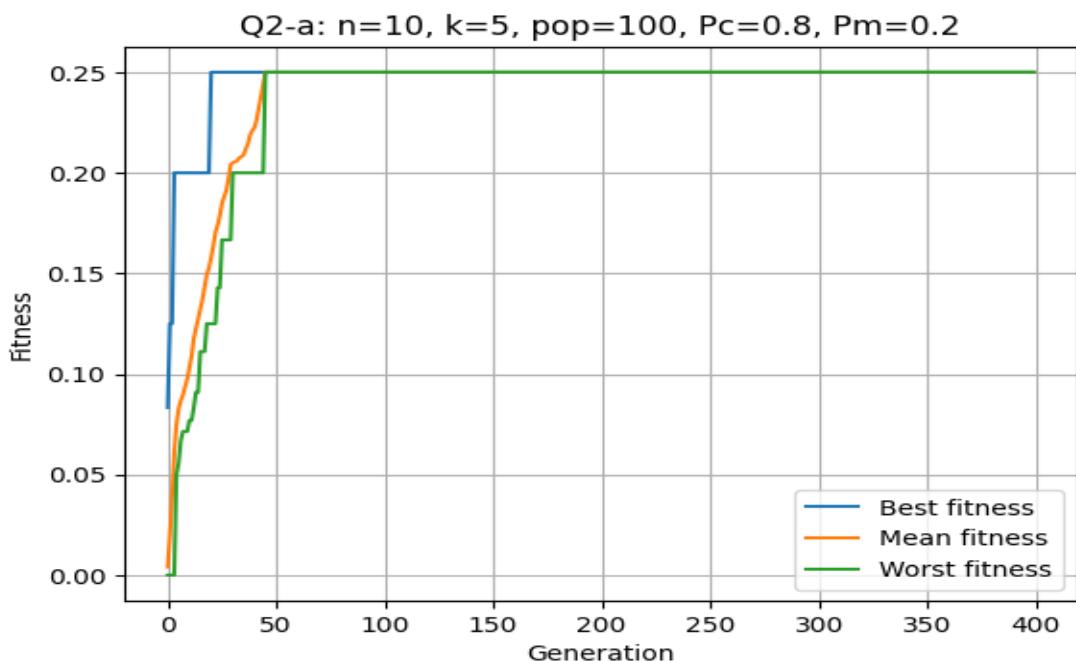
عملگرها؟

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 برآندگی (roulette) و در بخش اختیاری، tournament.

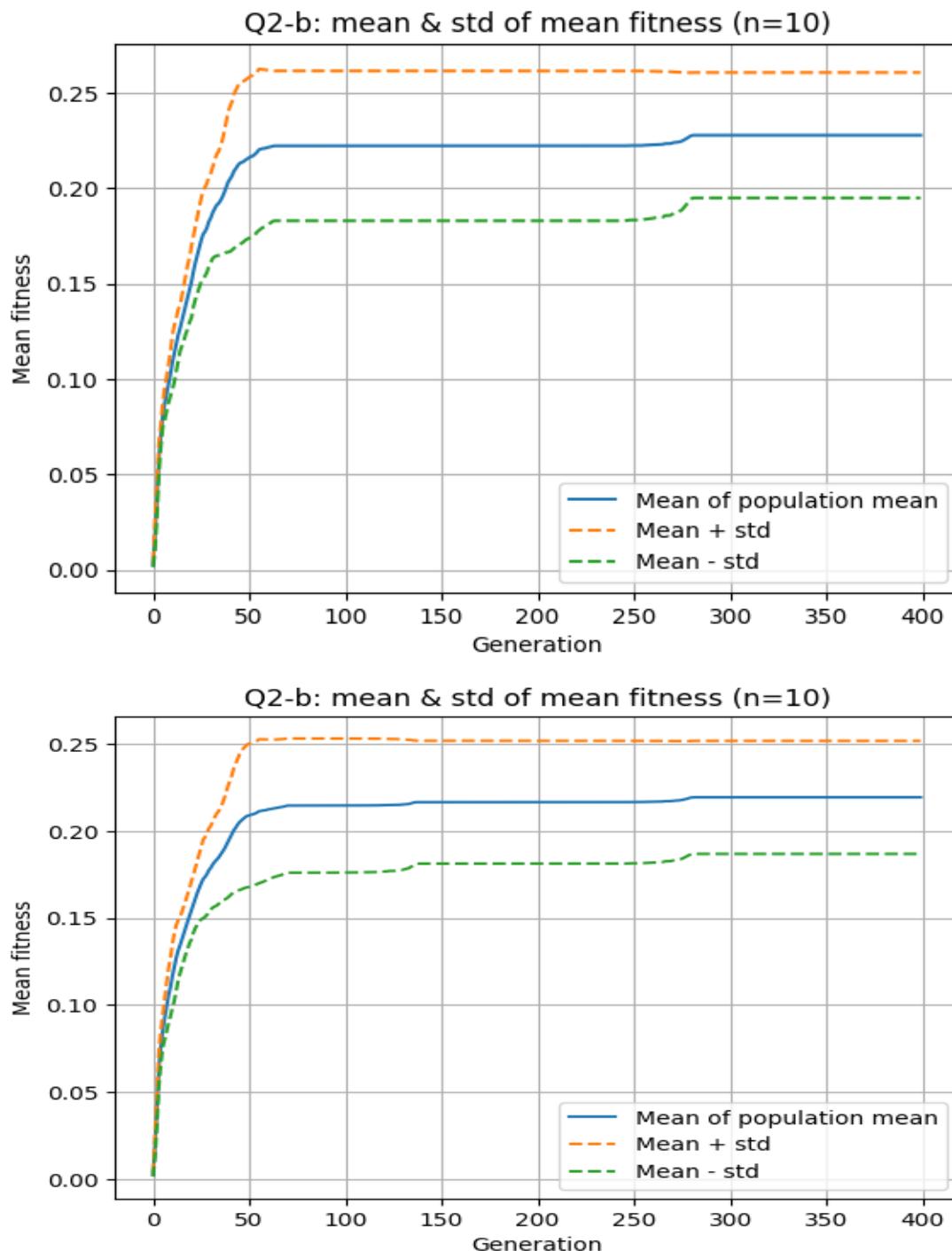
(الف)

برای $n=10$ و $k=5$ ، نمودار تکامل نشان داد که: plateau در چند ده نسل اول سریعاً بالا می‌روند و بعد به یک Mean و Best می‌رسند؛ به صورت تدریجی افزایش می‌یابد، که نشان‌دهنده‌ی حذف تدریجی تخصیص‌های Worst خیلی بد است.



(ب)

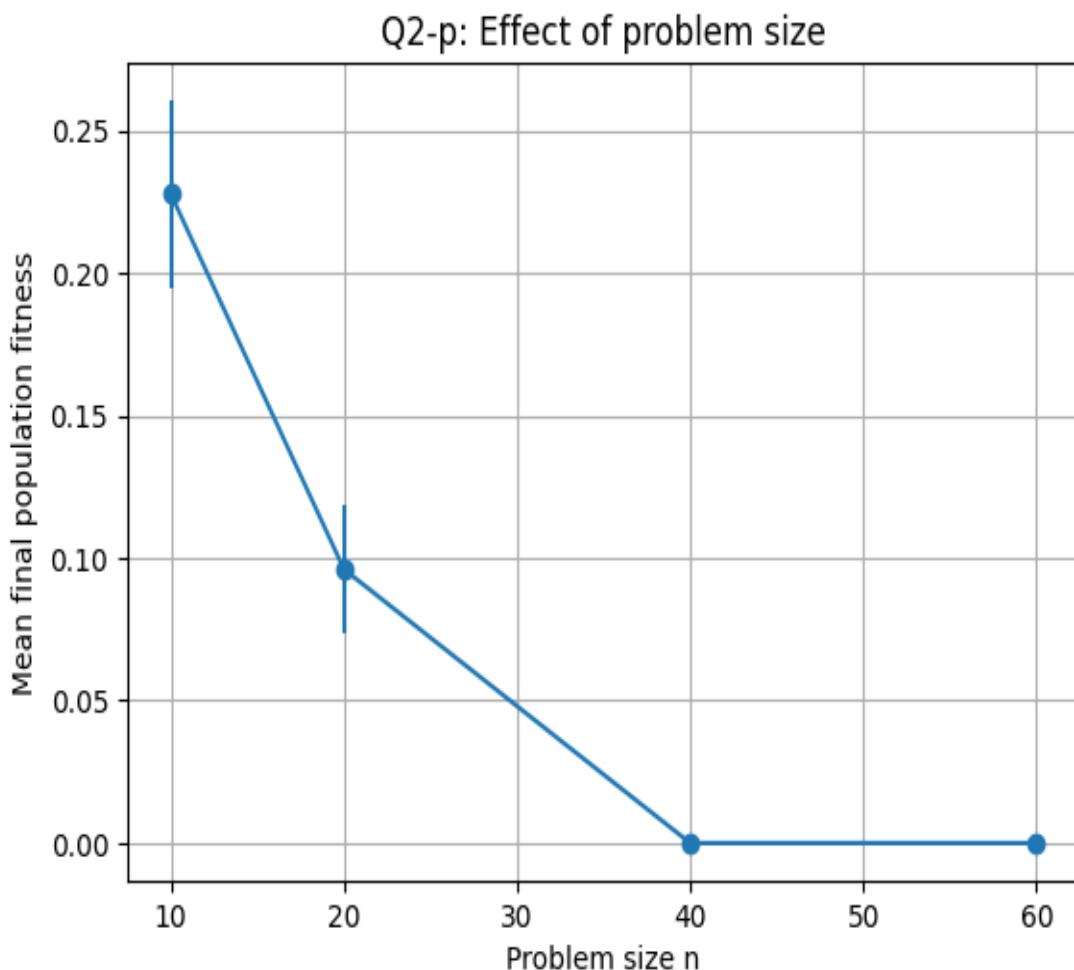
در ۶ اجرای مستقل (و ۱۲ بار)، میانگین برازنده‌گی نهایی در چند اجرای مختلف نزدیک هم است ولی انحراف معیار با افزایش n زیادتر می‌شود.



(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 برازنده‌گی نهایی به سمت مقادیر نزدیک صفر می‌رود؛
- از دید چشم‌انداز برازنده‌گی، landscape پیچیده‌تر و چندقله‌ای‌تر می‌شود، و EA برای رسیدن به راه حل‌های نزدیک به ایده‌آل به نسل‌ها و جمعیت‌های بسیار بزرگ‌تری نیاز دارد.

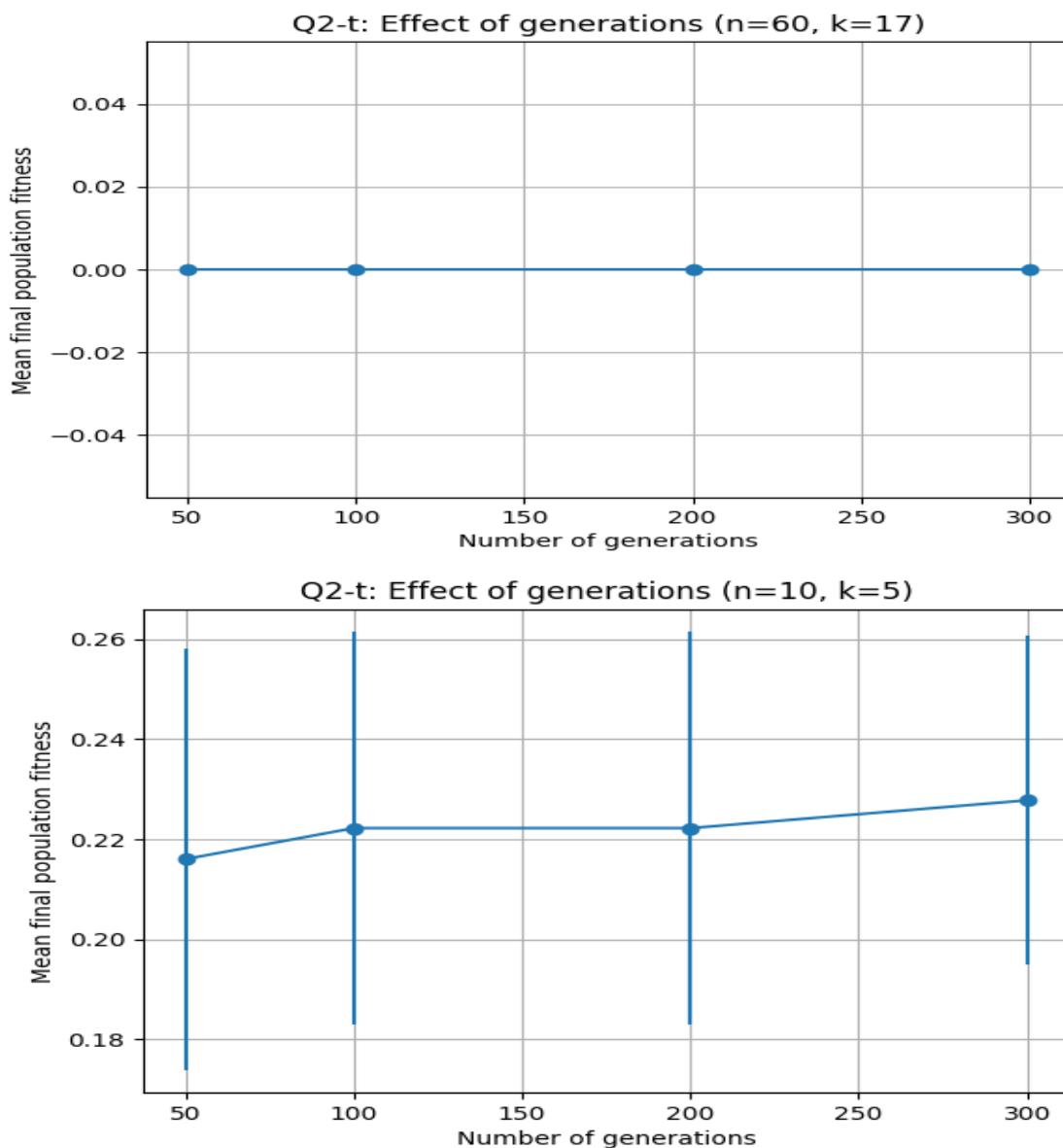


(ت)

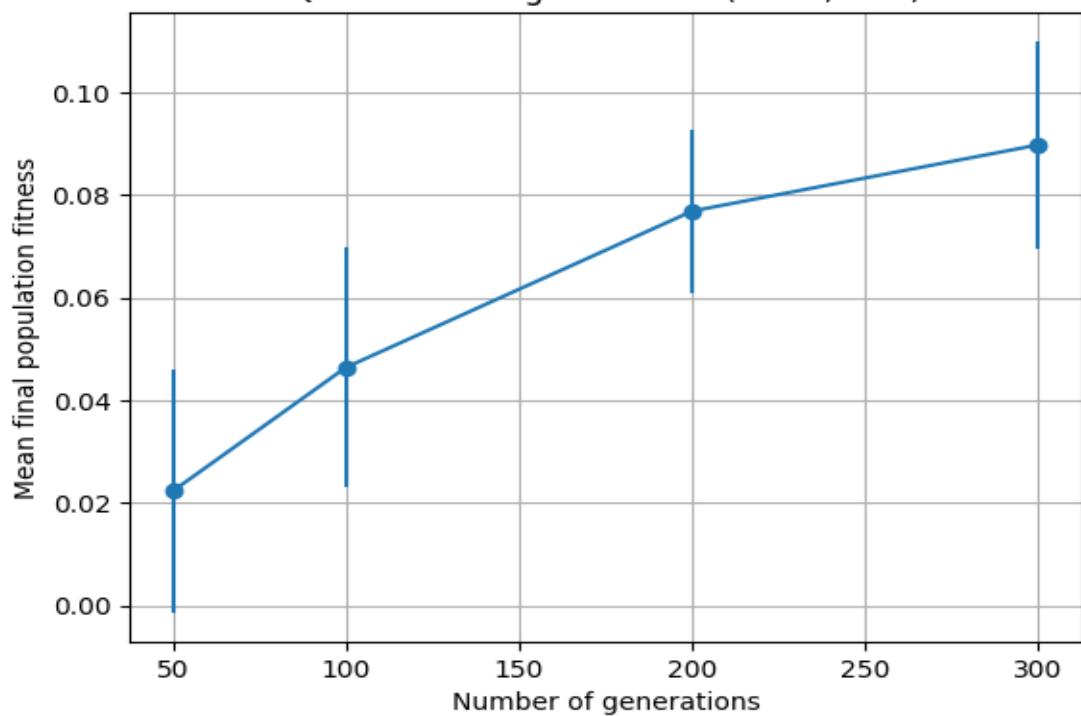
اثر تعداد نسل‌ها:

- در n کوچک، افزایش نسل احتمالاً تا حدی مفید است و به بهبود میانگین کمک می‌کند؛

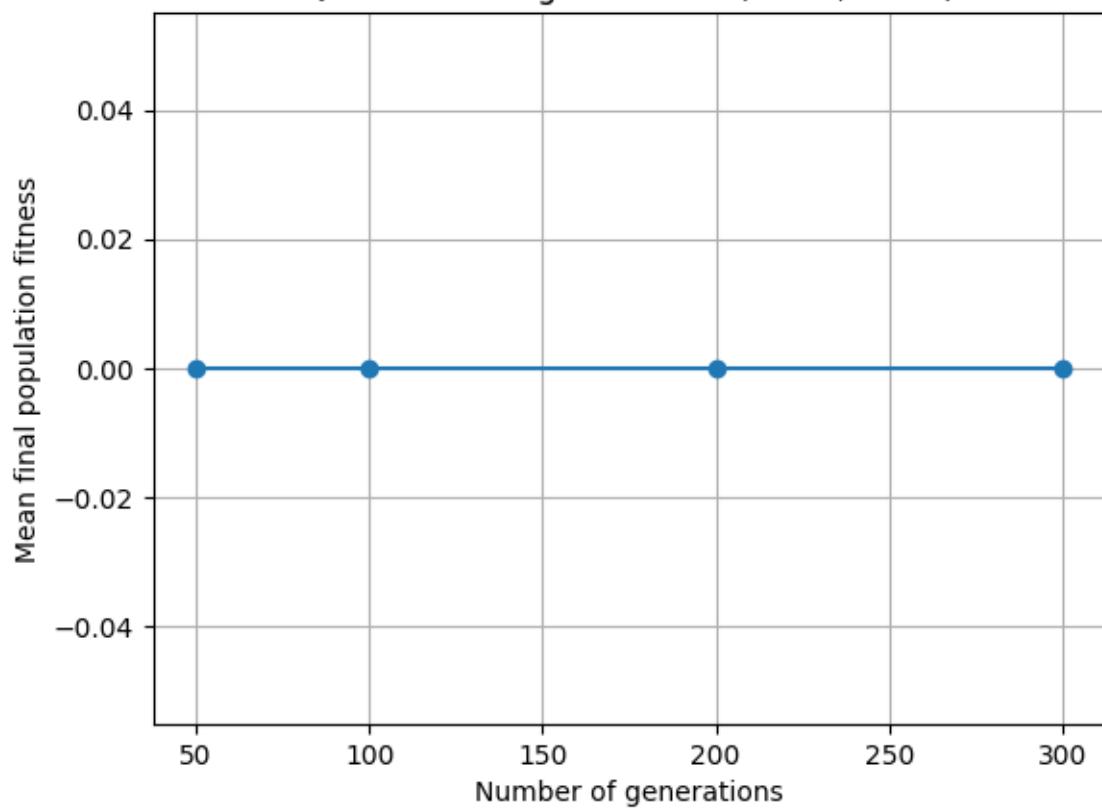
- در n ‌های بزرگ‌تر، بعد از یک حد مشخص، منحنی‌های میانگین تقریباً ثابت می‌شوند؛ یعنی الگوریتم وارد حالت exploitation شده و تنوع جمعیت کاهش یافته است.



Q2-t: Effect of generations ($n=20$, $k=8$)



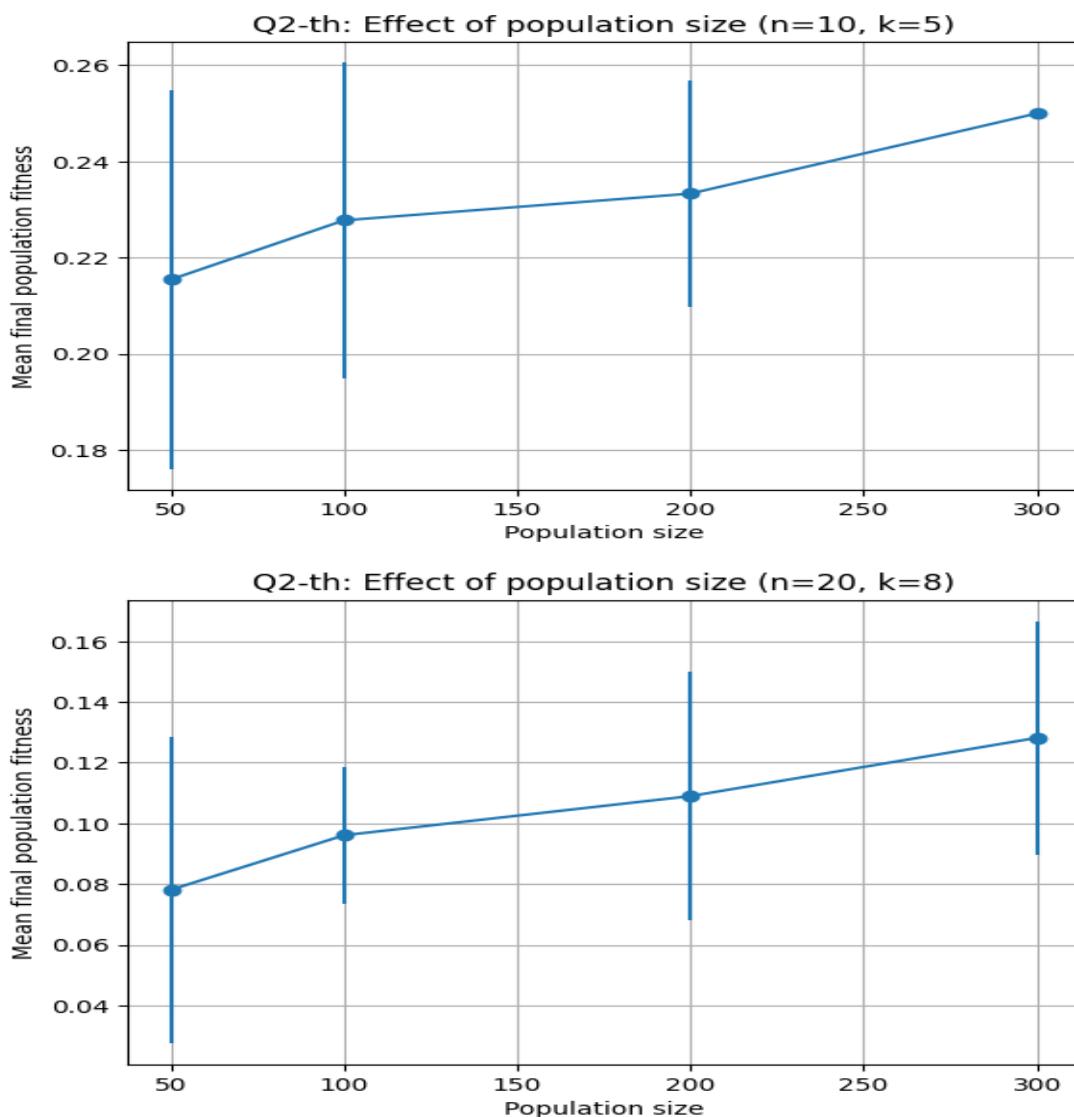
Q2-t: Effect of generations ($n=40$, $k=12$)



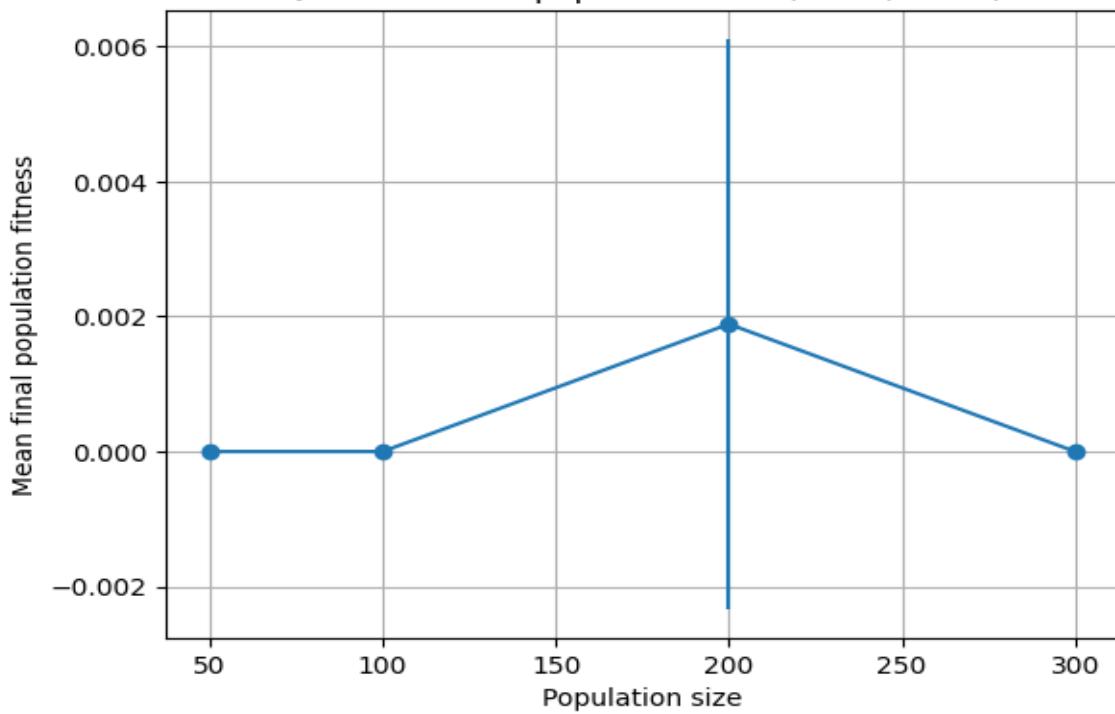
(ث)

اثر اندازه‌ی جمعیت:

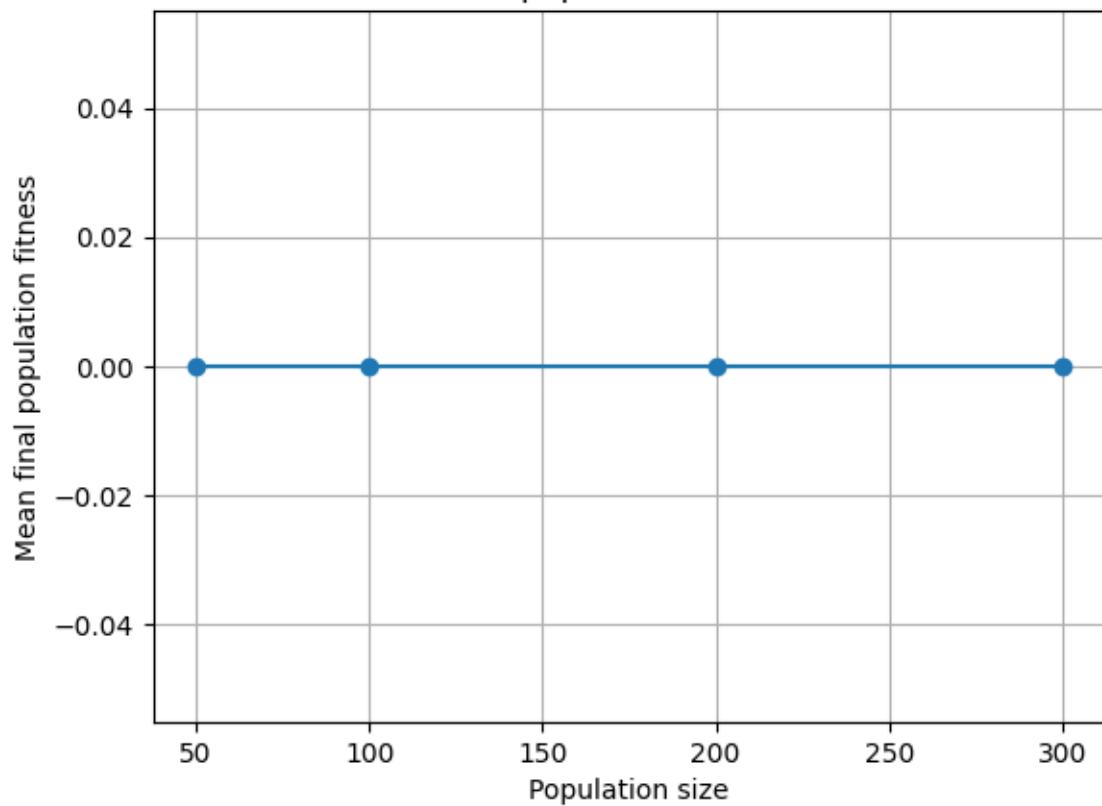
Pop کوچک یعنی الگوریتم در برخی اجراهای غیرمتعادل و با جریمه‌ی زیاد گیر می‌کند، و انحراف معیار بین اجراهای بالا می‌رود؛
 Pop بزرگ یعنی تنوع اولیه‌ی عالی، اما هزینه‌ی محاسباتی زیاد؛ منطبق با مفاهیم درس، جمعیت بزرگ فشار انتخاب را نسبتاً کاهش داده و اجازه می‌دهد افراد متوسط هم مدتی در جمعیت بمانند و تنوع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



Q2-th: Effect of population size ($n=40$, $k=12$)



Q2-th: Effect of population size ($n=60$, $k=17$)

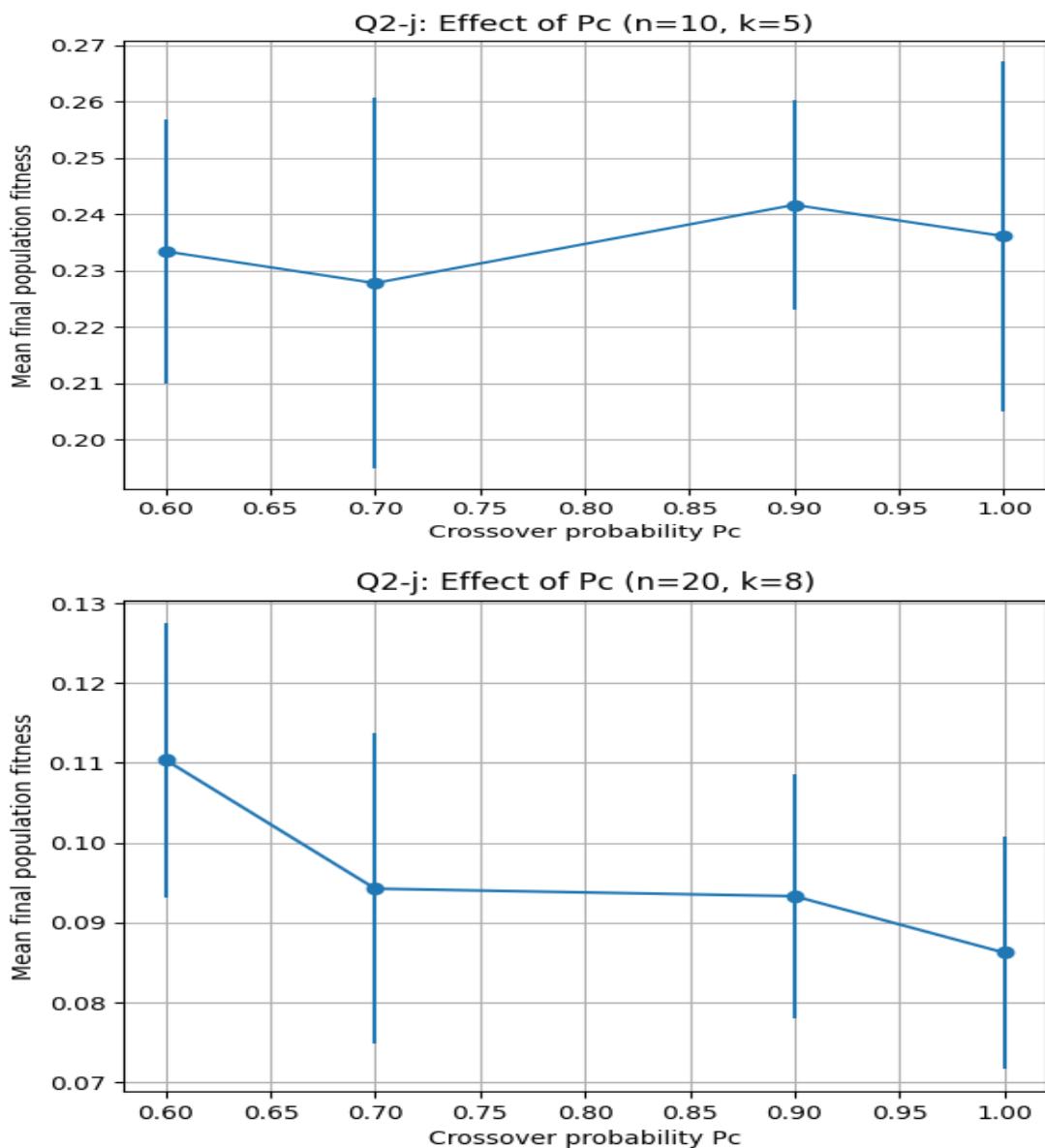


(ج)

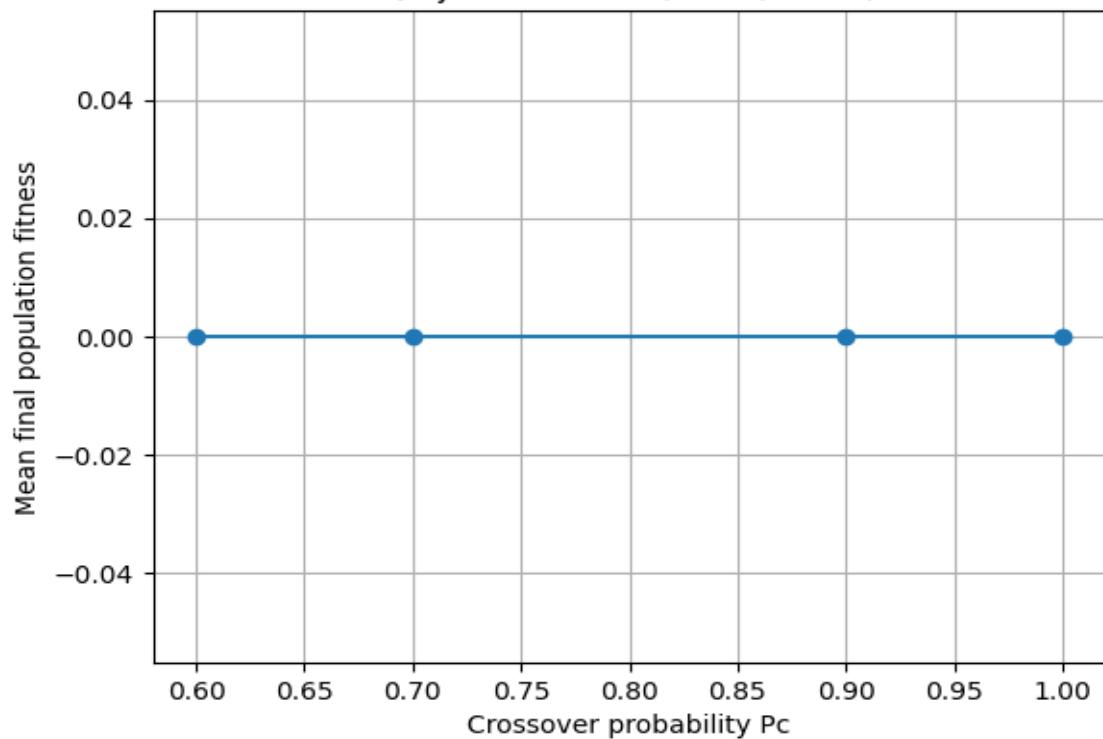
: P_c اثر

P_c خیلی پایین یعنی الگوریتم شبیه random walk با جهش می‌شود و استفاده‌ی مناسبی از recombination ندارد؛

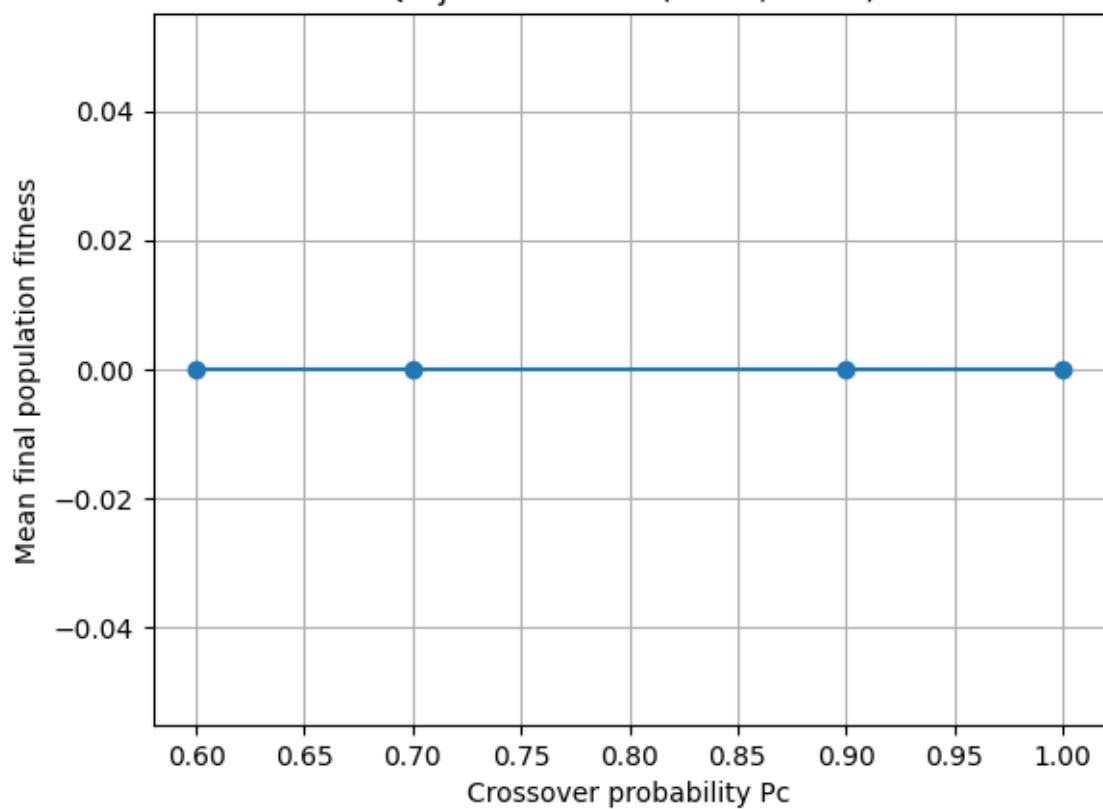
P_c خیلی بالا (نزدیک ۱) یعنی جابجایی قطعه‌های بزرگ از والدین، خوب است اما در حضور تنوع کم ممکن است تنها ترکیب چند الگوی مشابه باشد؛



Q2-j: Effect of P_c ($n=40, k=12$)



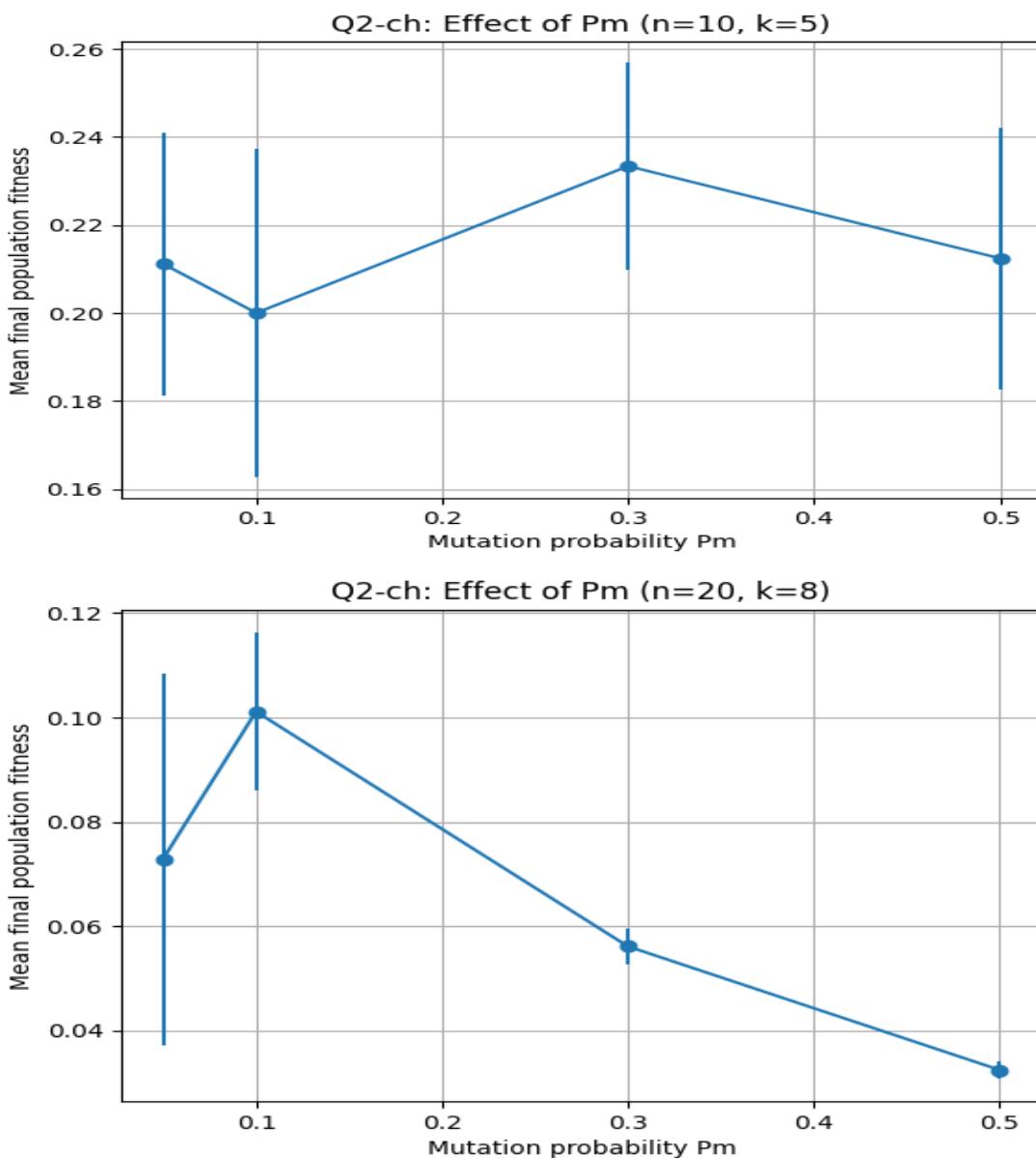
Q2-j: Effect of P_c ($n=60, k=17$)



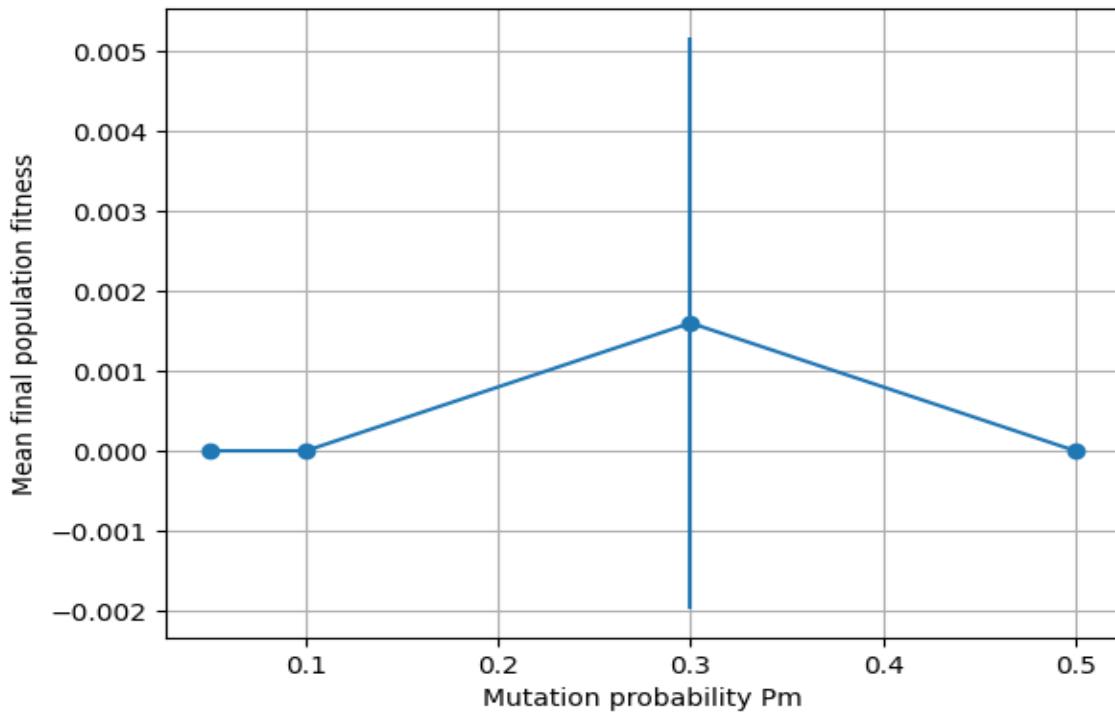
(ج)

: P_m اثر

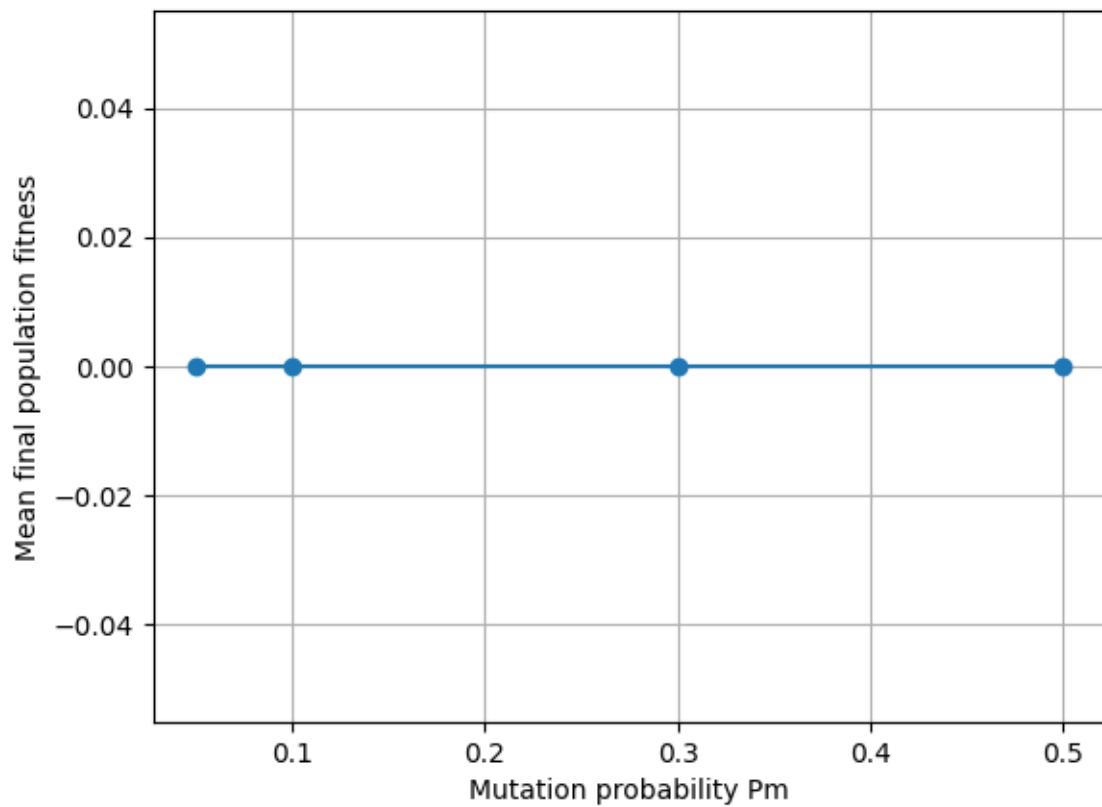
P_m کوچک یعنی سختی در فرار از تخصیص‌های گیر افتاده در قید سخت؛
 P_m بزرگ یعنی تخصیص‌ها دائمً دچار تغییر؛ امکان بالا نس دقیق بین گروه‌ها کاهش می‌یابد.



Q2-ch: Effect of Pm ($n=40$, $k=12$)



Q2-ch: Effect of Pm ($n=60$, $k=17$)



(ج)

بهترین راه حل ها:

- برای هر ترکیب پارامتر، باتابع `get_best_solution_info` بهترین `assignment`

در CSV ثبت شده، شامل:

○ بردار گروهها مثلًا؛ ۱,۲,۱,۳,۴,... :

○ جمع هر گروه؛

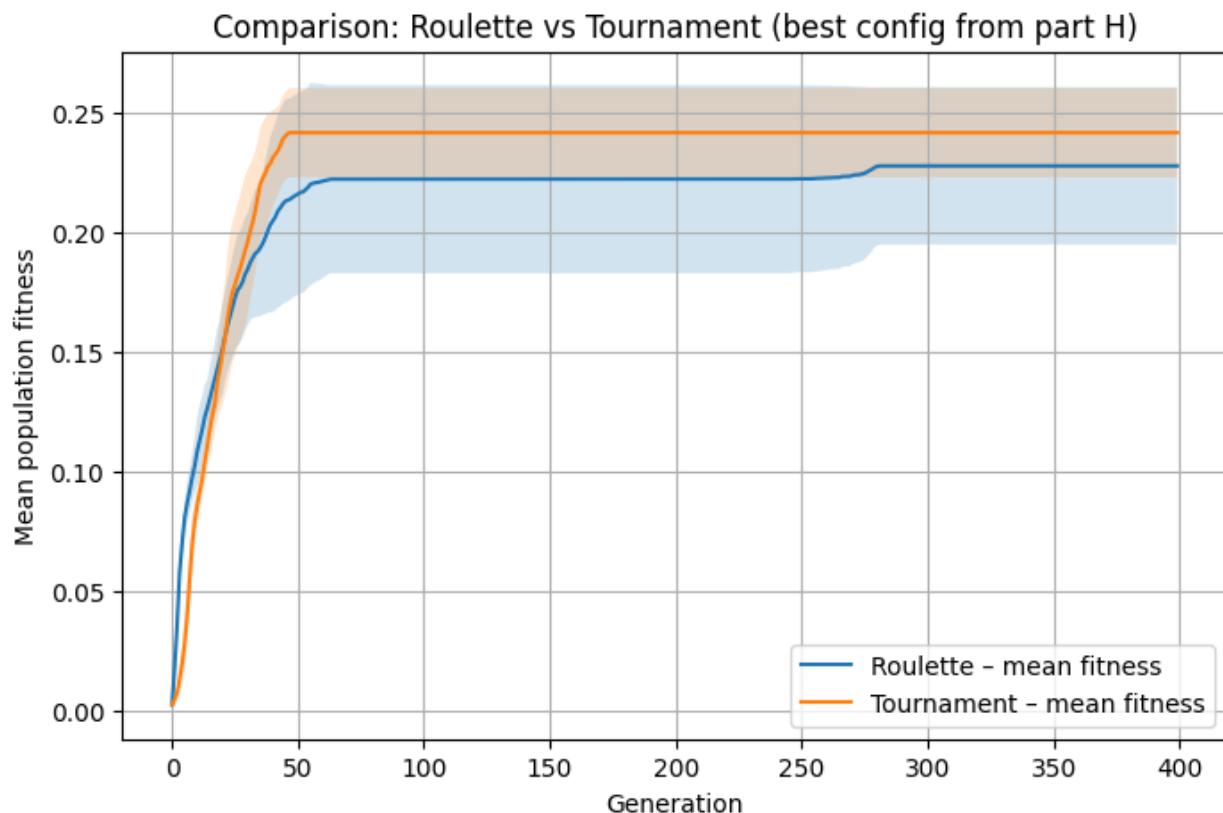
○ مقدار `fitness` و `penalty`

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	label	description	selection	n	k	pop_size	max_gene	Pc	Pm	best_run	best_fitness	assignment	group_sums		
2	a_base_n:Q2-a/b	b roulette		10	5	100	400	0.8	0.2	0	0.25	141531	g1: 10; g2: 10; g3: 13; g4: 11; g5: 11		
3	p_probler	Q2-p: effe roulette		10	5	100	400	0.8	0.2	0	0.25	141531	g1: 10; g2: 10; g3: 13; g4: 11; g5: 11		
4	p_probler	Q2-p: effe roulette		20	8	100	400	0.8	0.2	1	0.142857	657135	g1: 23; g2: 29; g3: 25; g4: 28; g5: 23;		
5	p_probler	Q2-p: effe roulette		40	12	100	400	0.8	0.2	0	0.614124	g1: 63; g2: 75; g3: 43; g4: 222; g5: 38;			
6	p_probler	Q2-p: effe roulette		60	17	100	400	0.8	0.2	0	0.1316144	g1: 207; g2: 130; g3: 34; g4: 169; g5: 1			
7	t_generat	Q2-t: effe roulette		10	5	100	50	0.8	0.2	0	0.25	141531	g1: 10; g2: 10; g3: 13; g4: 11; g5: 11		
8	t_generat	Q2-t: effe roulette		10	5	100	100	0.8	0.2	0	0.25	141531	g1: 10; g2: 10; g3: 13; g4: 11; g5: 11		
9	t_generat	Q2-t: effe roulette		10	5	100	200	0.8	0.2	0	0.25	141531	g1: 10; g2: 10; g3: 13; g4: 11; g5: 11		
10	t_generat	Q2-t: effe roulette		10	5	100	300	0.8	0.2	0	0.25	141531	g1: 10; g2: 10; g3: 13; g4: 11; g5: 11		
11	t_generat	Q2-t: effe roulette		20	8	100	50	0.8	0.2	2	0.071429	628763	g1: 27; g2: 25; g3: 23; g4: 31; g5: 34;		
12	t_generat	Q2-t: effe roulette		20	8	100	100	0.8	0.2	1	0.125	746136	g1: 23; g2: 29; g3: 29; g4: 22; g5: 25;		
13	t_generat	Q2-t: effe roulette		20	8	100	200	0.8	0.2	1	0.142857	657135	g1: 23; g2: 29; g3: 25; g4: 28; g5: 23;		
14	t_generat	Q2-t: effe roulette		20	8	100	300	0.8	0.2	1	0.142857	657135	g1: 23; g2: 29; g3: 25; g4: 28; g5: 23;		
15	t_generat	Q2-t: effe roulette		40	12	100	50	0.8	0.2	0	0.614124	g1: 63; g2: 75; g3: 43; g4: 222; g5: 38;			
16	t_generat	Q2-t: effe roulette		40	12	100	100	0.8	0.2	0	0.614124	g1: 63; g2: 75; g3: 43; g4: 222; g5: 38;			
17	t_generat	Q2-t: effe roulette		40	12	100	200	0.8	0.2	0	0.614124	g1: 63; g2: 75; g3: 43; g4: 222; g5: 38;			
18	t_generat	Q2-t: effe roulette		40	12	100	300	0.8	0.2	0	0.614124	g1: 63; g2: 75; g3: 43; g4: 222; g5: 38;			
19	t_generat	Q2-t: effe roulette		60	17	100	50	0.8	0.2	0	0.1316144	g1: 207; g2: 130; g3: 34; g4: 169; g5: 1			
20	t_generat	Q2-t: effe roulette		60	17	100	100	0.8	0.2	0	0.1316144	g1: 207; g2: 130; g3: 34; g4: 169; g5: 1			
21	t_generat	Q2-t: effe roulette		60	17	100	200	0.8	0.2	0	0.1316144	g1: 207; g2: 130; g3: 34; g4: 169; g5: 1			
22	t_generat	Q2-t: effe roulette		60	17	100	300	0.8	0.2	0	0.1316144	g1: 207; g2: 130; g3: 34; g4: 169; g5: 1			
23	th_pop_n:Q2-th:	eff roulette		10	5	50	400	0.8	0.2	0	0.25	242352	g1: 10; g2: 10; g3: 11; g4: 11; g5: 13		

(خ)

مقایسه‌ی انتخاب roulette با tournament نشان داد:

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



۴- بخش ۳: مسئله چند وزیر با نمایش جایگشت

نمایش و تابع برازنده‌گی؟

- نمایش: جایگشت طول n ، که سطراها 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.

عملگرهای؟

- 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 جایه‌جایی یک وزیر در ستون‌های مختلف است.
- انتخاب: متناسب با برازنده‌گی 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 $n = 9$:
- 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 ۸ وزیر و ۹ وزیر برای تنظیمات انتخاب شده ساده‌تر از آن چیزی است که فکر می‌کردیم و EA در اکثر تنظیم‌ها جواب کامل را پیدا می‌کند.
- برای $n = 10$ و $n = 11$:

 - 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 n های کوچک، حتی ۵۰ نسل کافی است؛ افزایش بیشتر فقط زمان را زیاد می‌کند؛
 - برای n های بزرگ‌تر، ۲۰۰-۱۰۰ 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 ۶,۰ 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 (مثلًا ۱,۳—۰,۰) بهترین trade-off را ایجاد کردن.

نتایج نهایی

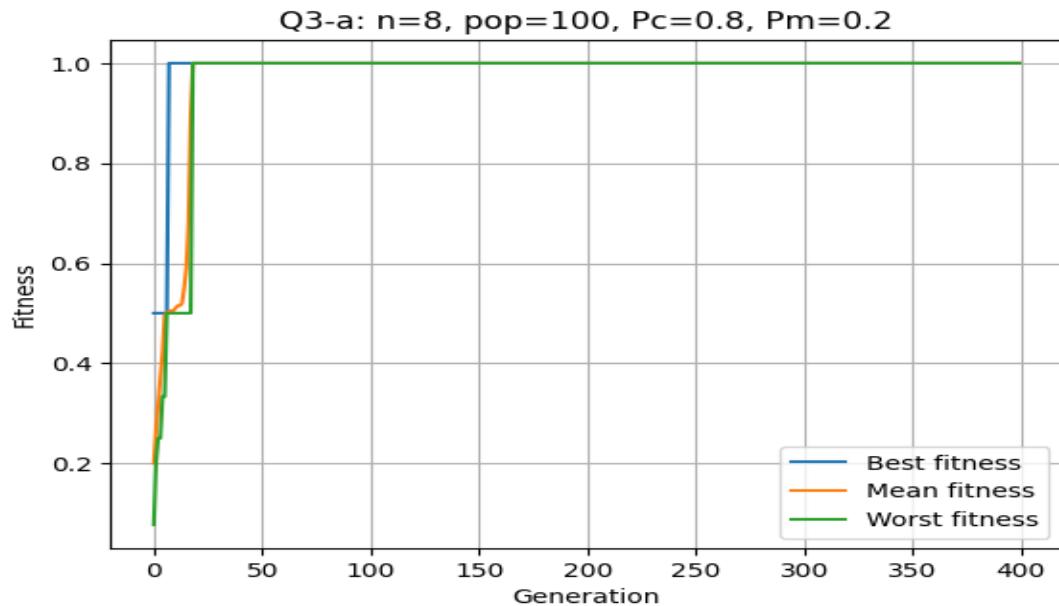
◦ مسئله‌ی n-Queens Landscape نسبتاً «نرم» است: تعداد زیادی جواب قابل قبول (بدون برخورد) وجود دارد، مخصوصاً برای n های کوچک؛

◦ selection pressure متوسط رولت به همراه CX و جهش درج برای پیدا کردن جواب کافی است؛

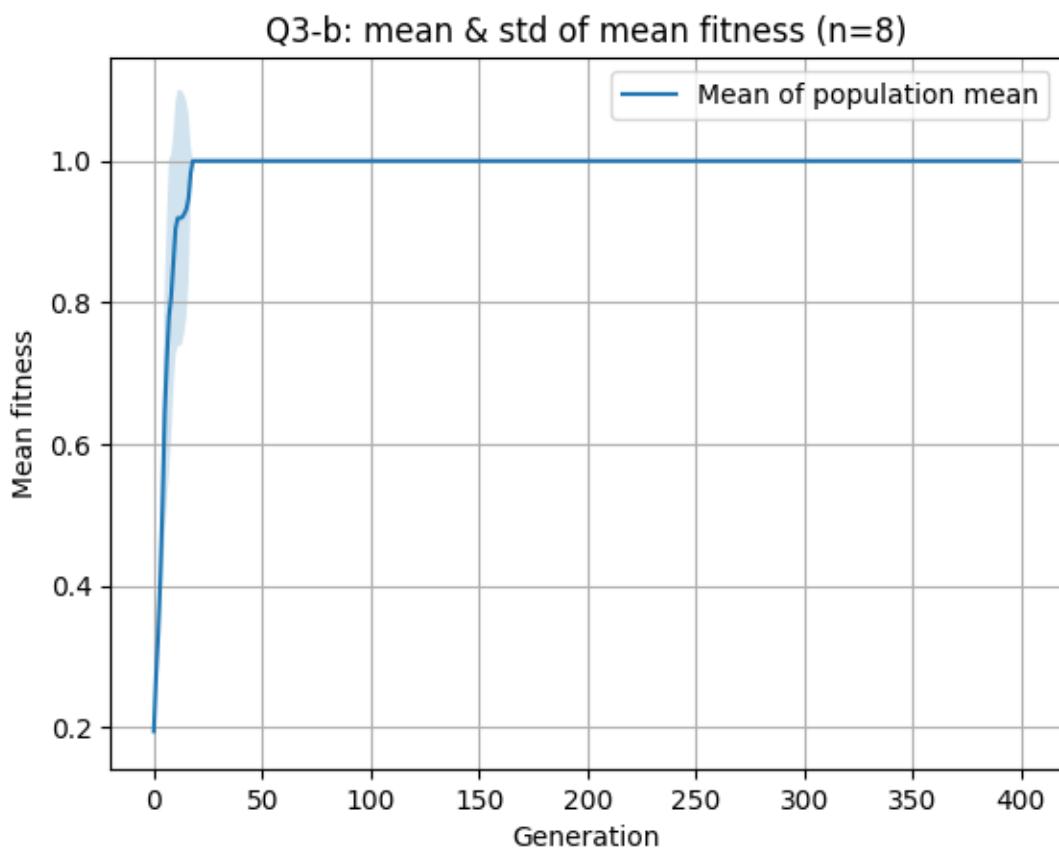
◦ برخلاف Rastrigin، این مسئله برای EA به صورت global یک مسئله‌ی نسبتاً راحت است؛ همین باعث می‌شود نمودارها اغلب به فیتنس یک برسند.

نمودارها نیز به صورت زیر اند:

(الف)

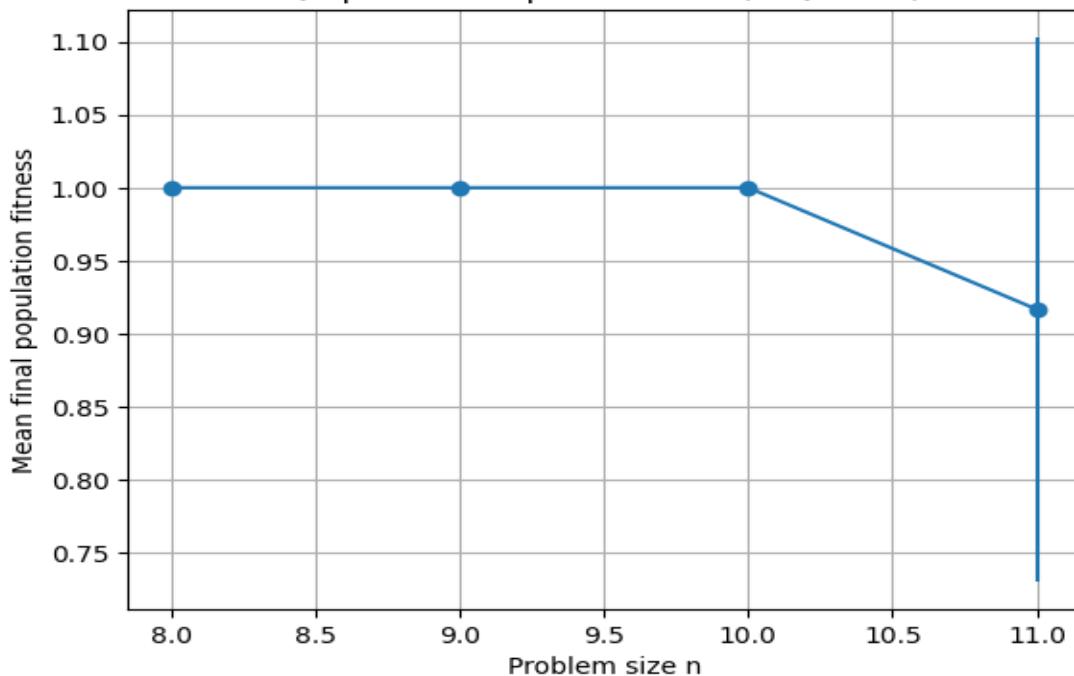


(ب)



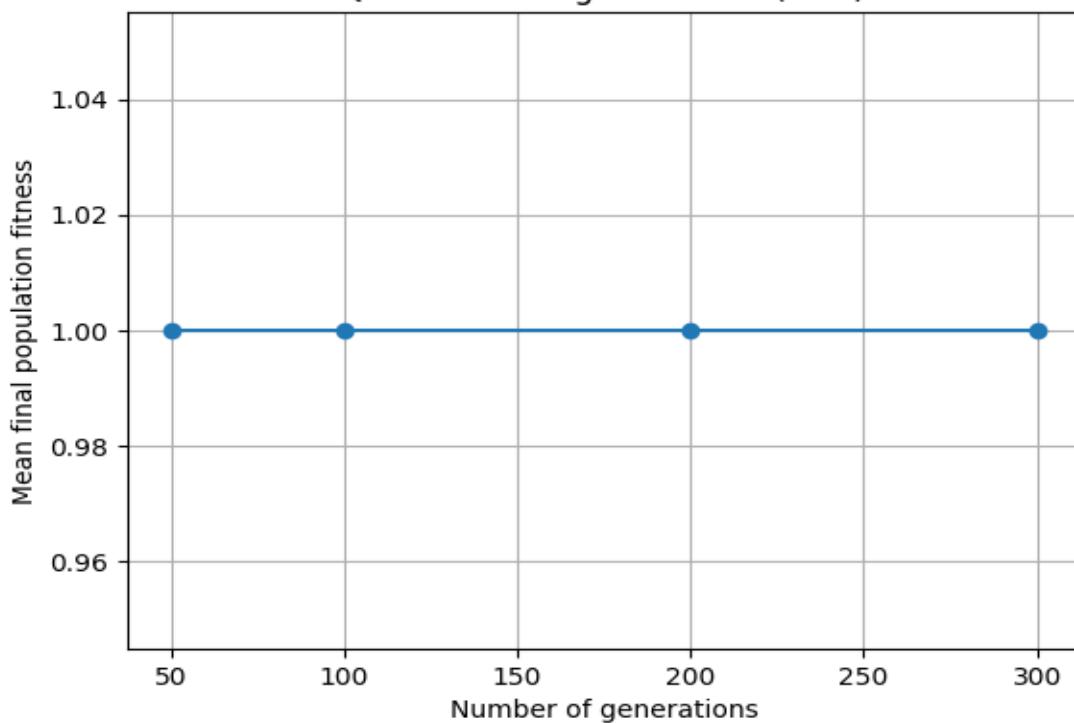
(پ)

Q3-p: Effect of problem size (N-Queens)

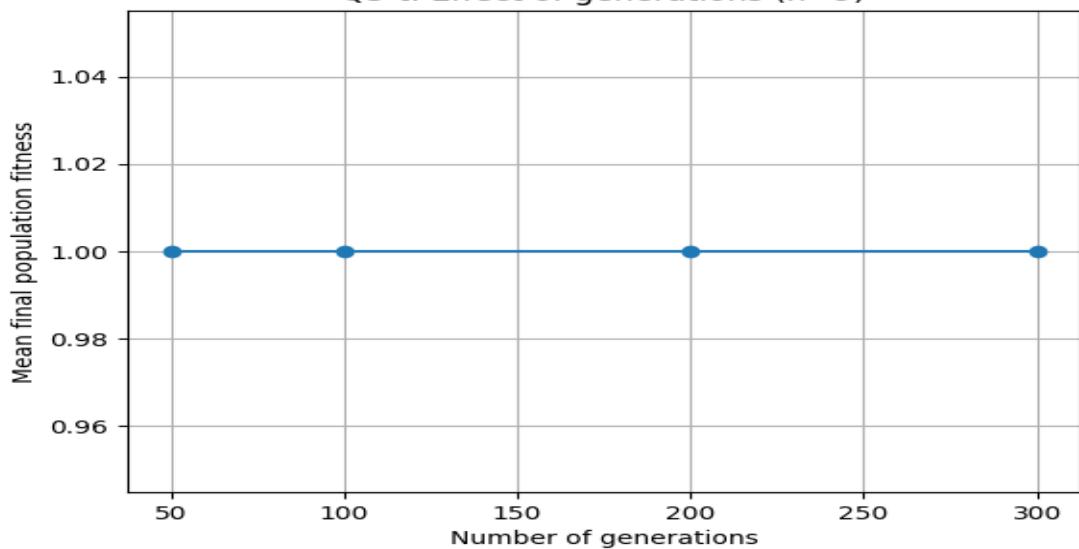


(ت)

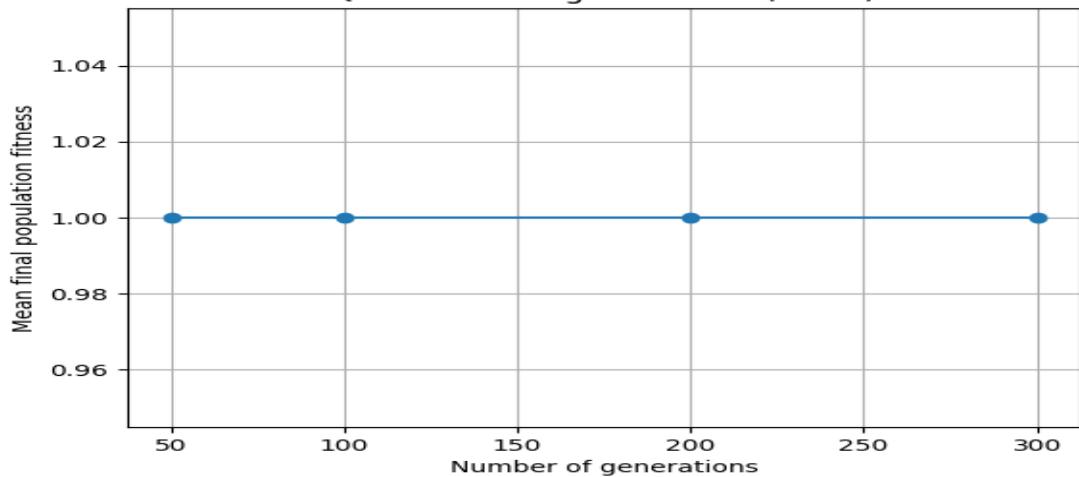
Q3-t: Effect of generations (n=8)



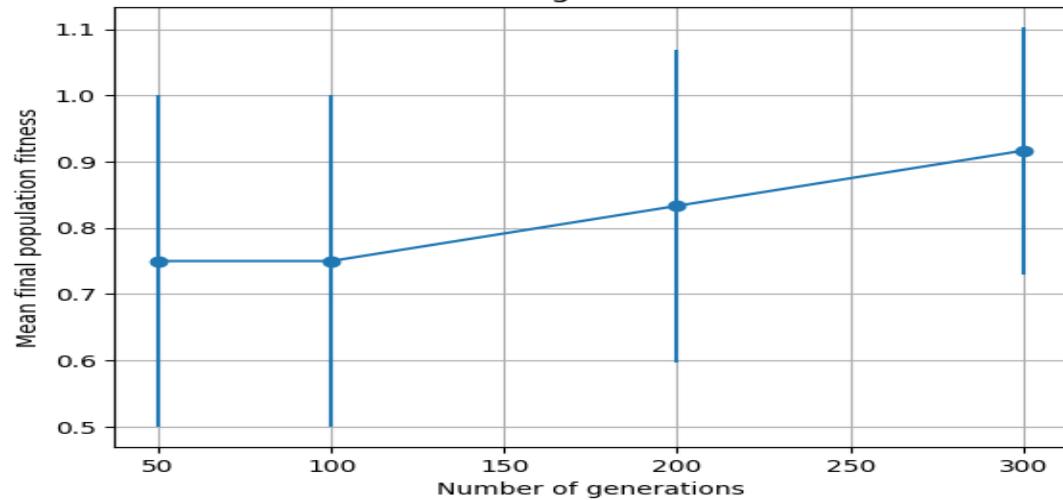
Q3-t: Effect of generations (n=9)



Q3-t: Effect of generations (n=10)

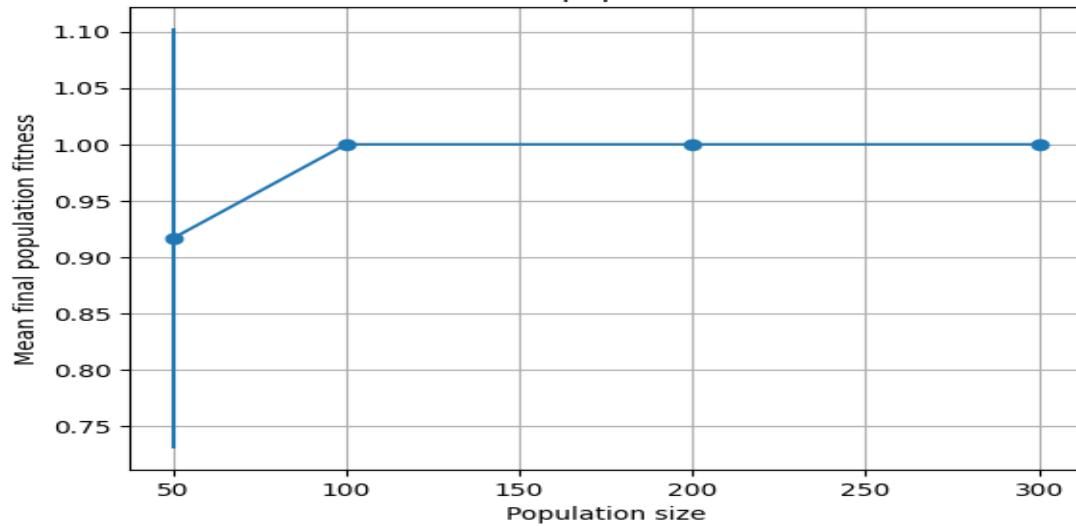


Q3-t: Effect of generations (n=11)

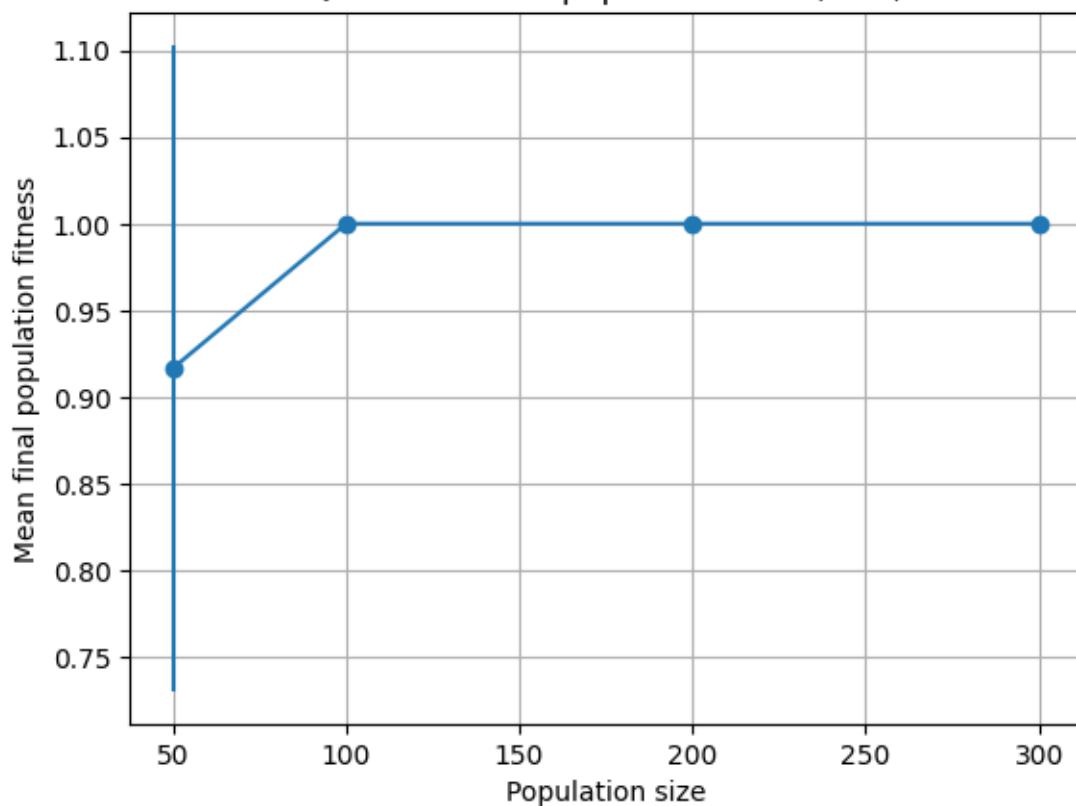


(*)

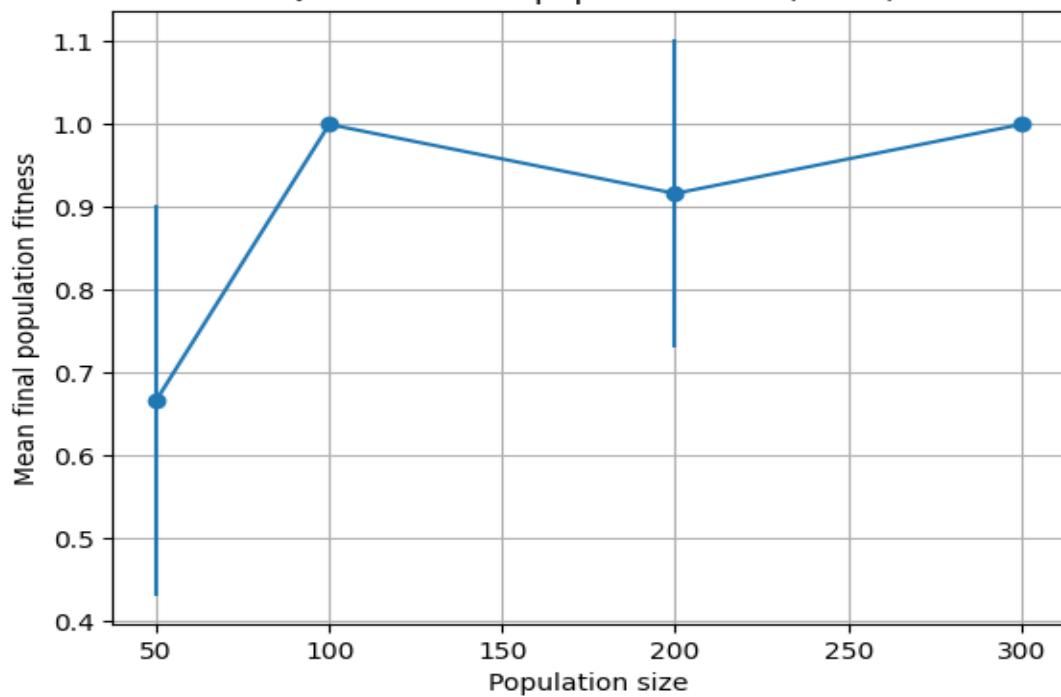
Q3-th: Effect of population size ($n=8$)



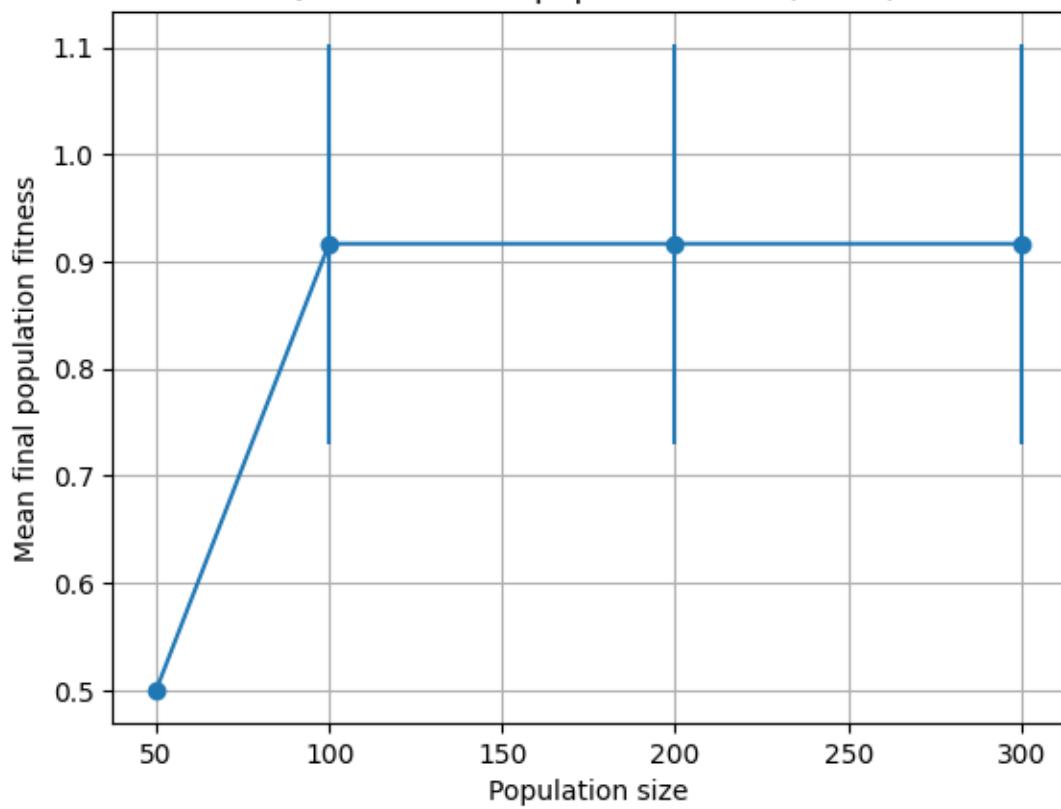
Q3-th: Effect of population size ($n=9$)



Q3-th: Effect of population size (n=10)

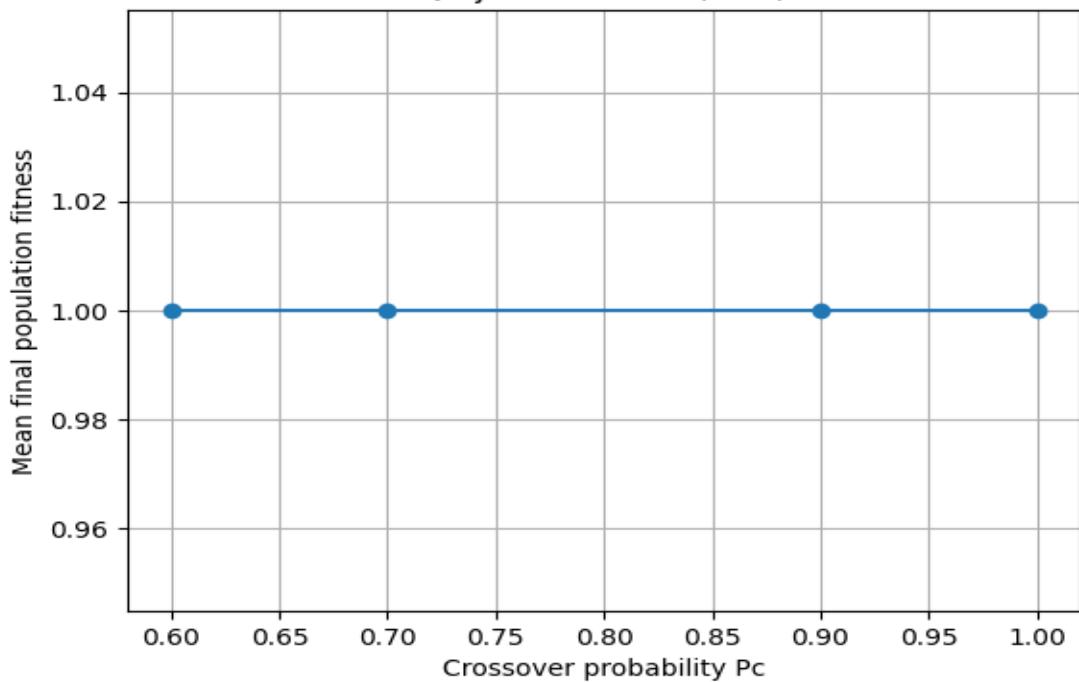


Q3-th: Effect of population size (n=11)

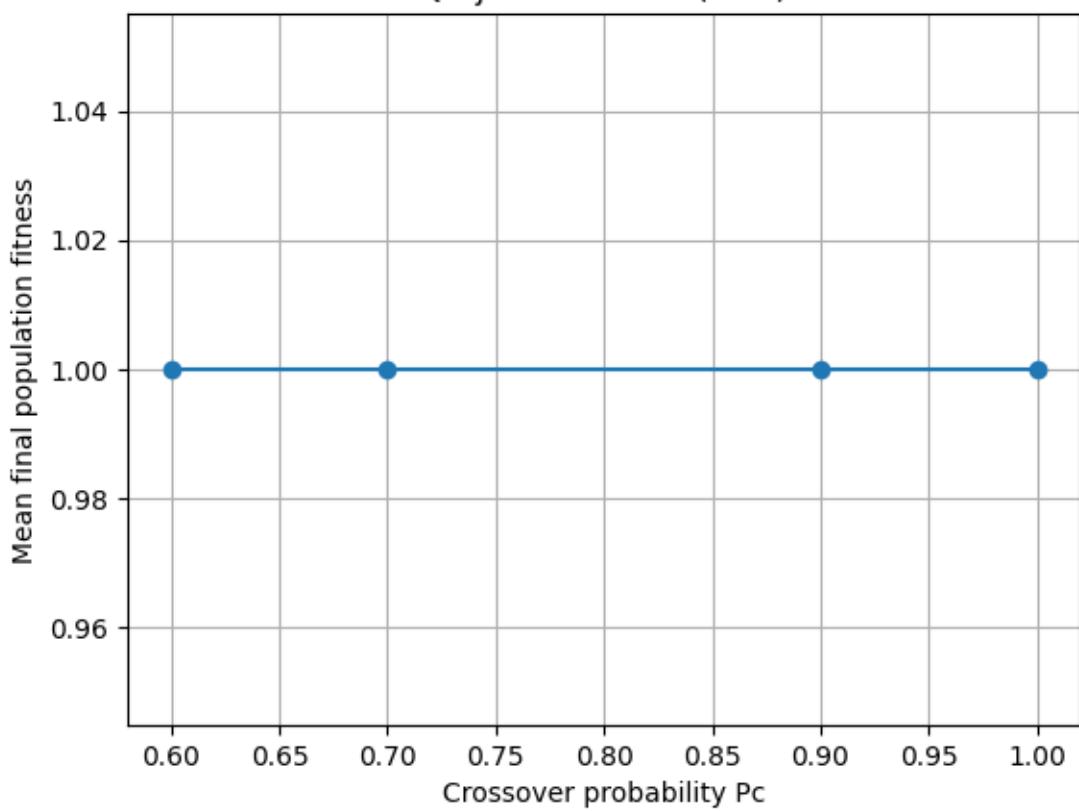


(c)

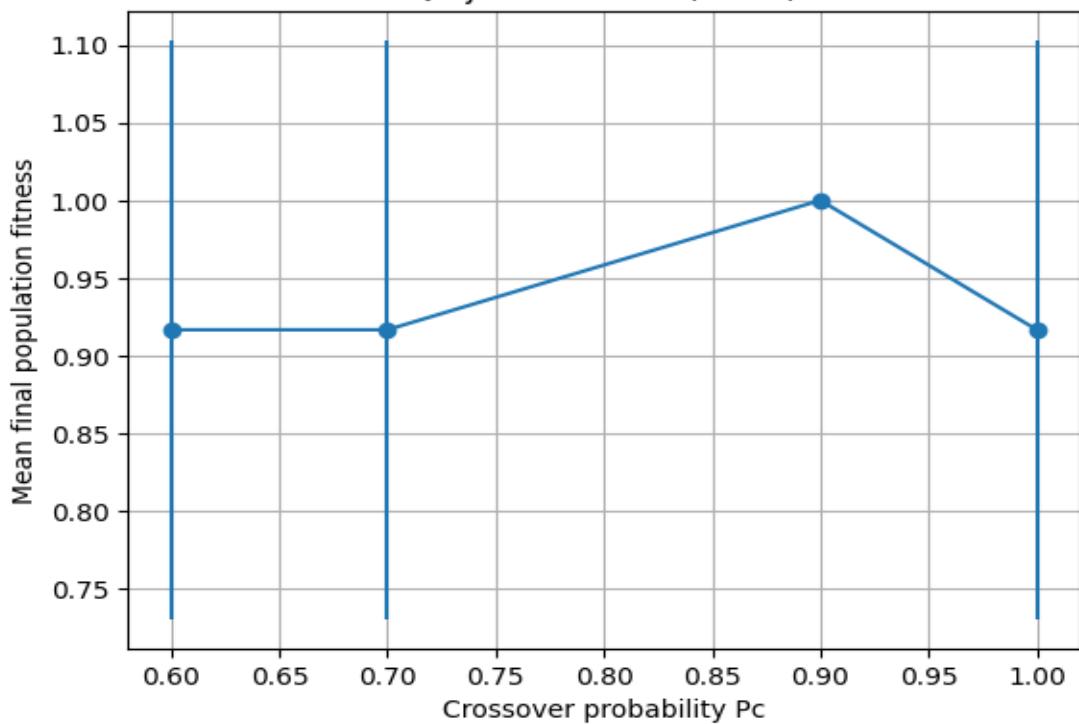
Q3-j: Effect of P_c ($n=8$)



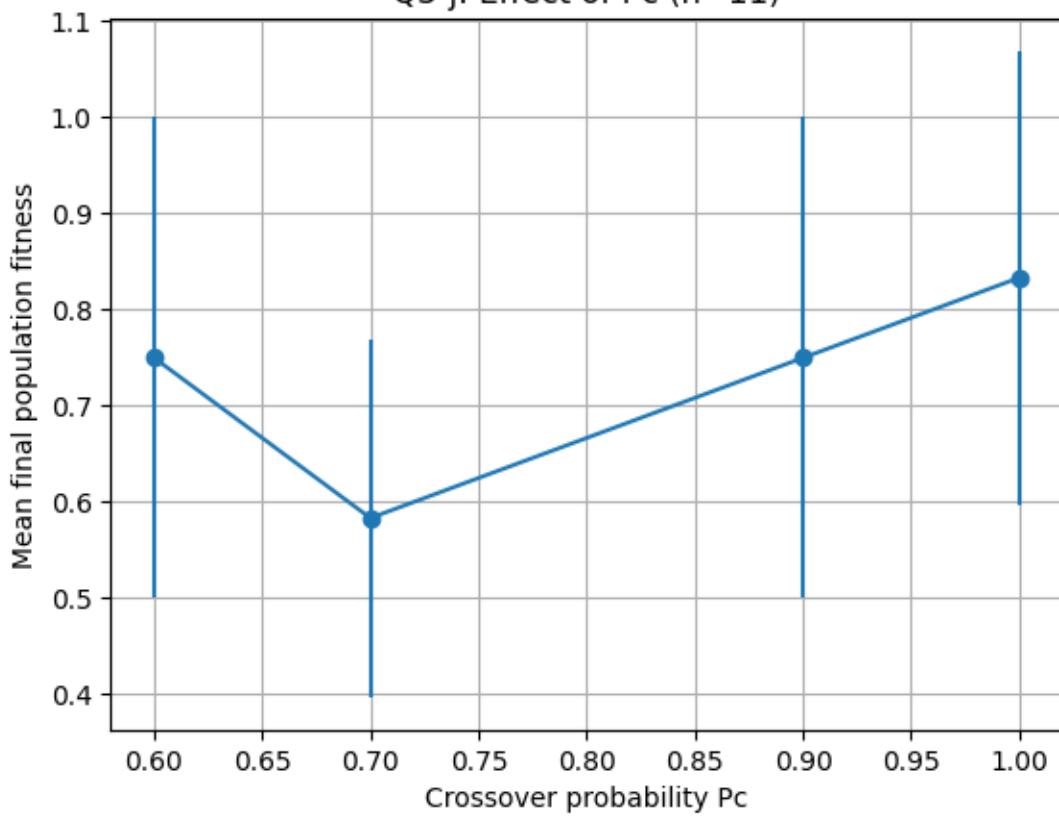
Q3-j: Effect of P_c ($n=9$)



Q3-j: Effect of P_c ($n=10$)

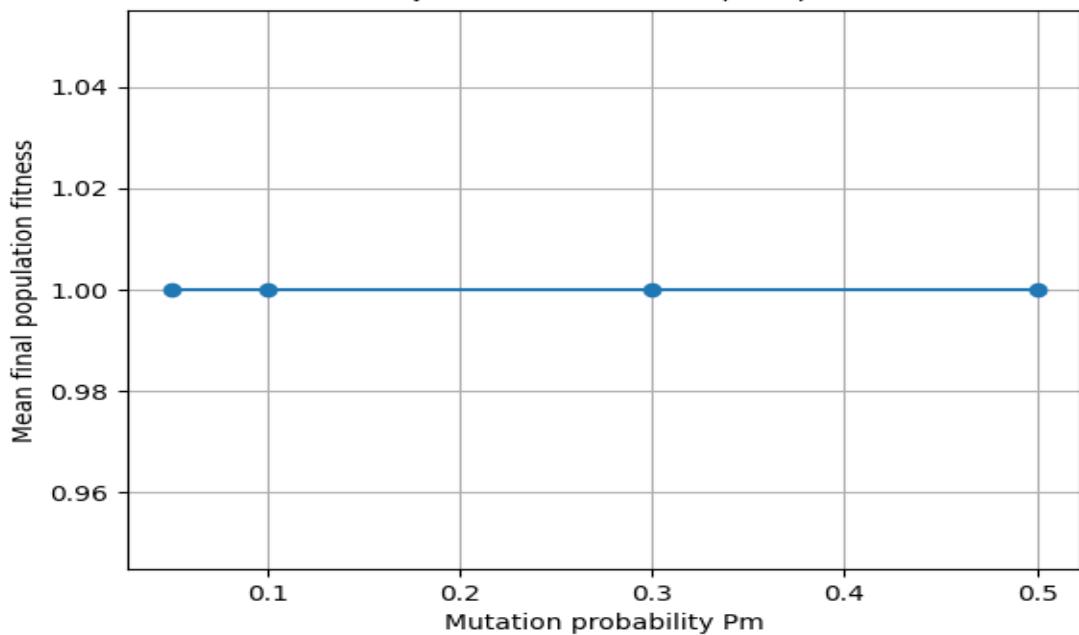


Q3-j: Effect of P_c ($n=11$)

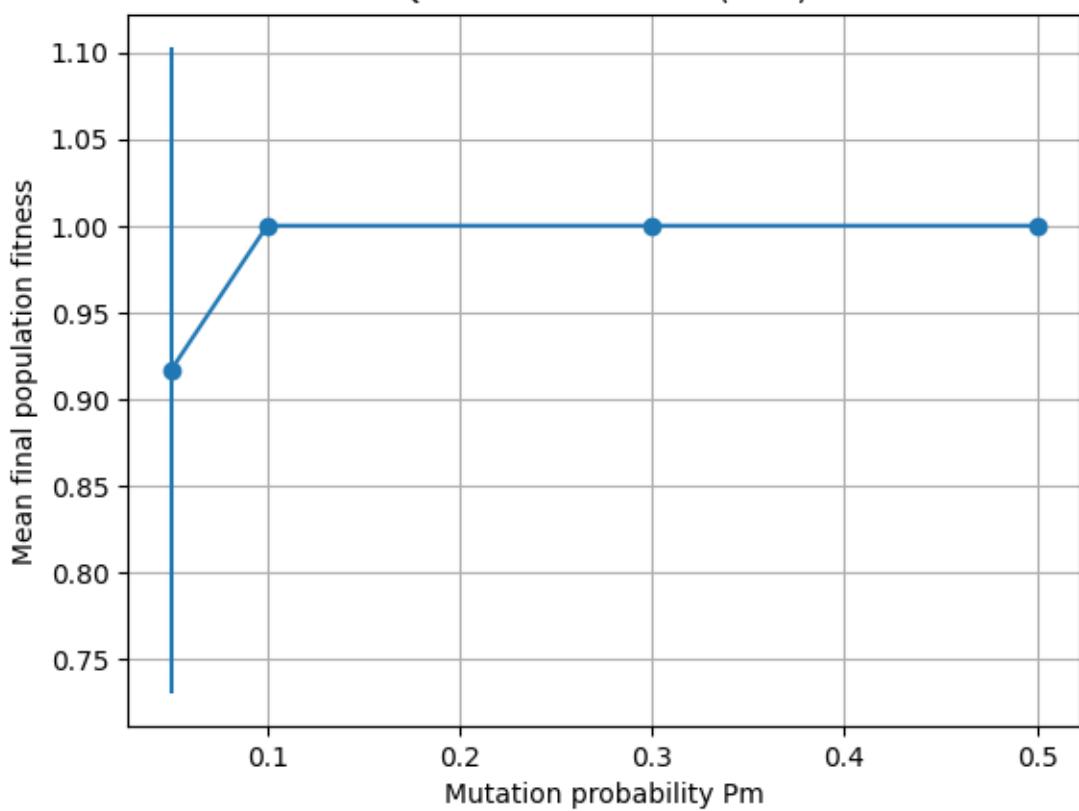


(c)

Q3-ch: Effect of Pm (n=8)

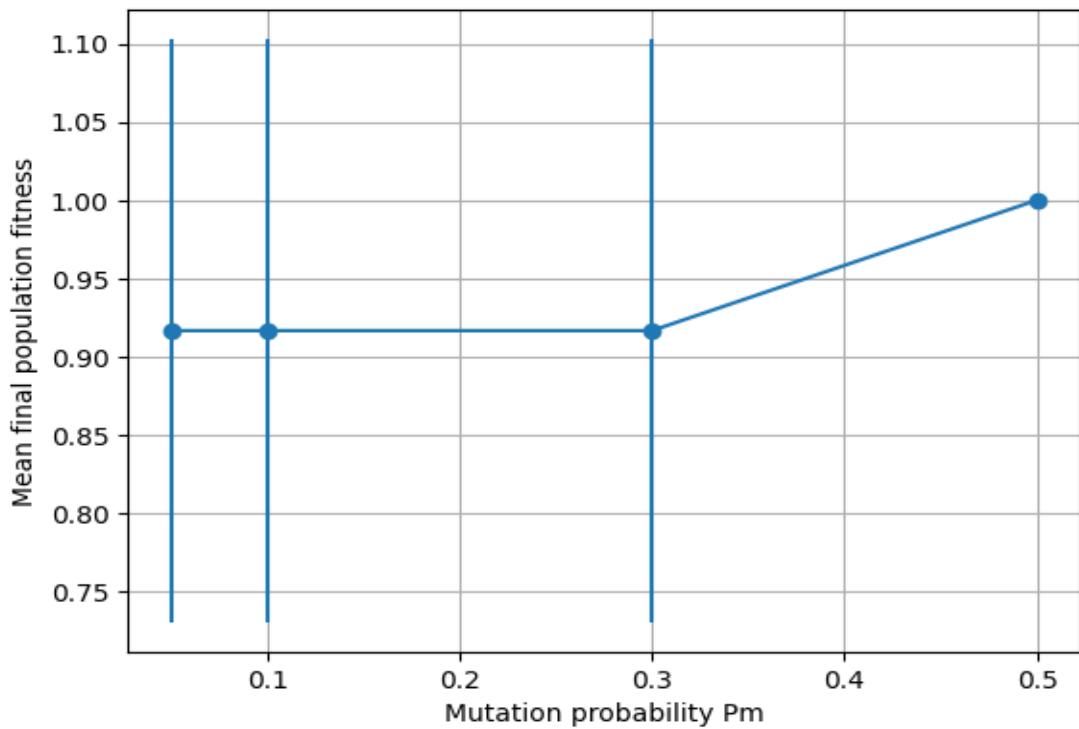


Q3-ch: Effect of Pm (n=9)

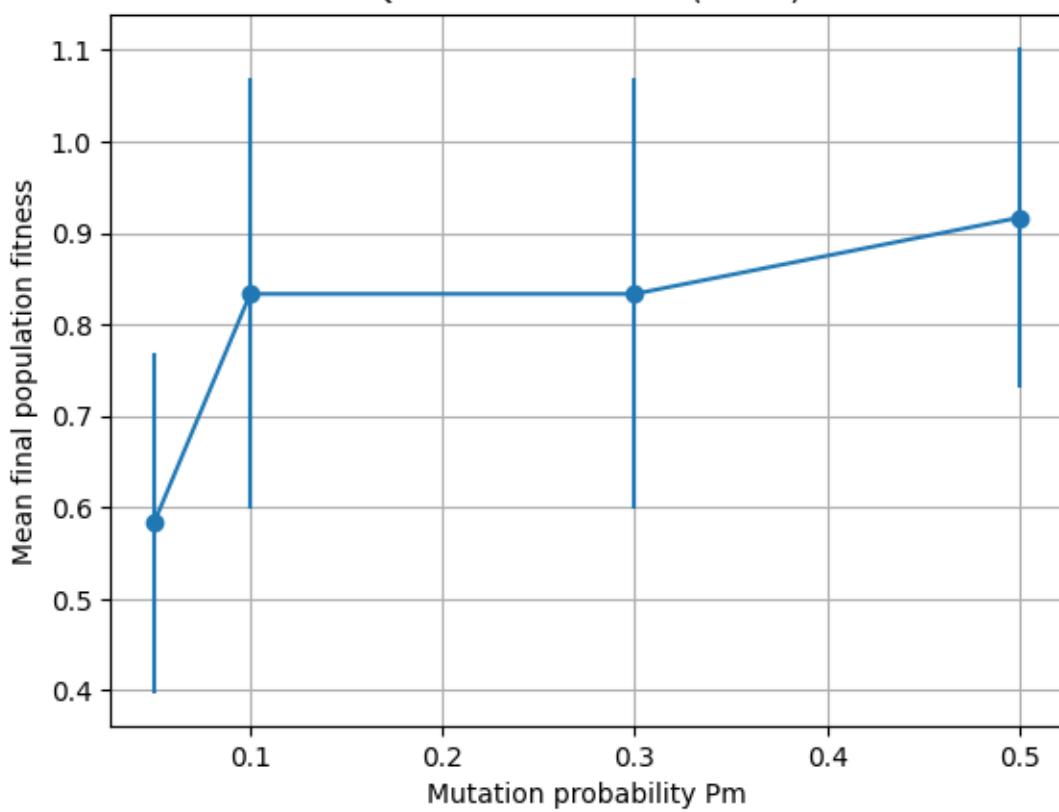


yy

Q3-ch: Effect of P_m ($n=10$)



Q3-ch: Effect of P_m ($n=11$)

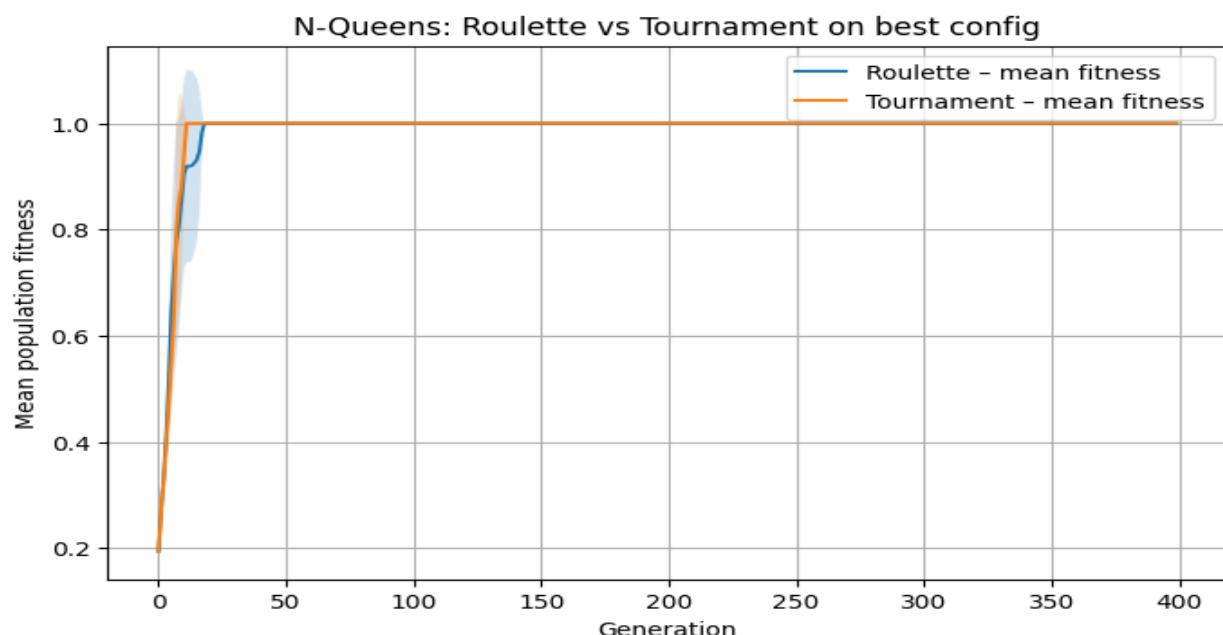


(ح)

Excel screenshot showing a CSV file named "EA_nqueens_best_solutions_all_configs.csv". The table has columns: question, n, pop_size, max_genePc, Pm, best_run, best_fitness, conflicts, perm. Rows show various configurations for an 8-queens problem.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	question	n	pop_size	max_genePc	Pm	best_run	best_fitness	conflicts	perm						
2	a_b_base_	8	100	400	0.8	0.2	0	1	004752613						
3	p_probler	8	100	400	0.8	0.2	0	1	004752613						
4	p_probler	9	100	400	0.8	0.2	0	1	0374205186						
5	p_probler	10	100	400	0.8	0.2	0	1	02586307149						
6	p_probler	11	100	400	0.8	0.2	0	1	0085210379461						
7	t_generat	8	100	50	0.8	0.2	0	1	004752613						
8	t_generat	8	100	100	0.8	0.2	0	1	004752613						
9	t_generat	8	100	200	0.8	0.2	0	1	004752613						
10	t_generat	8	100	300	0.8	0.2	0	1	004752613						
11	t_generat	9	100	50	0.8	0.2	0	1	0374205186						
12	t_generat	9	100	100	0.8	0.2	0	1	0374205186						
13	t_generat	9	100	200	0.8	0.2	0	1	0374205186						
14	t_generat	9	100	300	0.8	0.2	0	1	0374205186						
15	t_generat	10	100	50	0.8	0.2	0	1	02586307149						
16	t_generat	10	100	100	0.8	0.2	0	1	02586307149						
17	t_generat	10	100	200	0.8	0.2	0	1	02586307149						
18	t_generat	10	100	300	0.8	0.2	0	1	02586307149						
19	t_generat	11	100	50	0.8	0.2	1	1	0682039710415						
20	t_generat	11	100	100	0.8	0.2	1	1	0682039710415						
21	t_generat	11	100	200	0.8	0.2	1	1	0682039710415						
22	t_generat	11	100	300	0.8	0.2	0	1	0085210379461						
23	th_popula	8	50	400	0.8	0.2	0	1	025147063						

(خ)

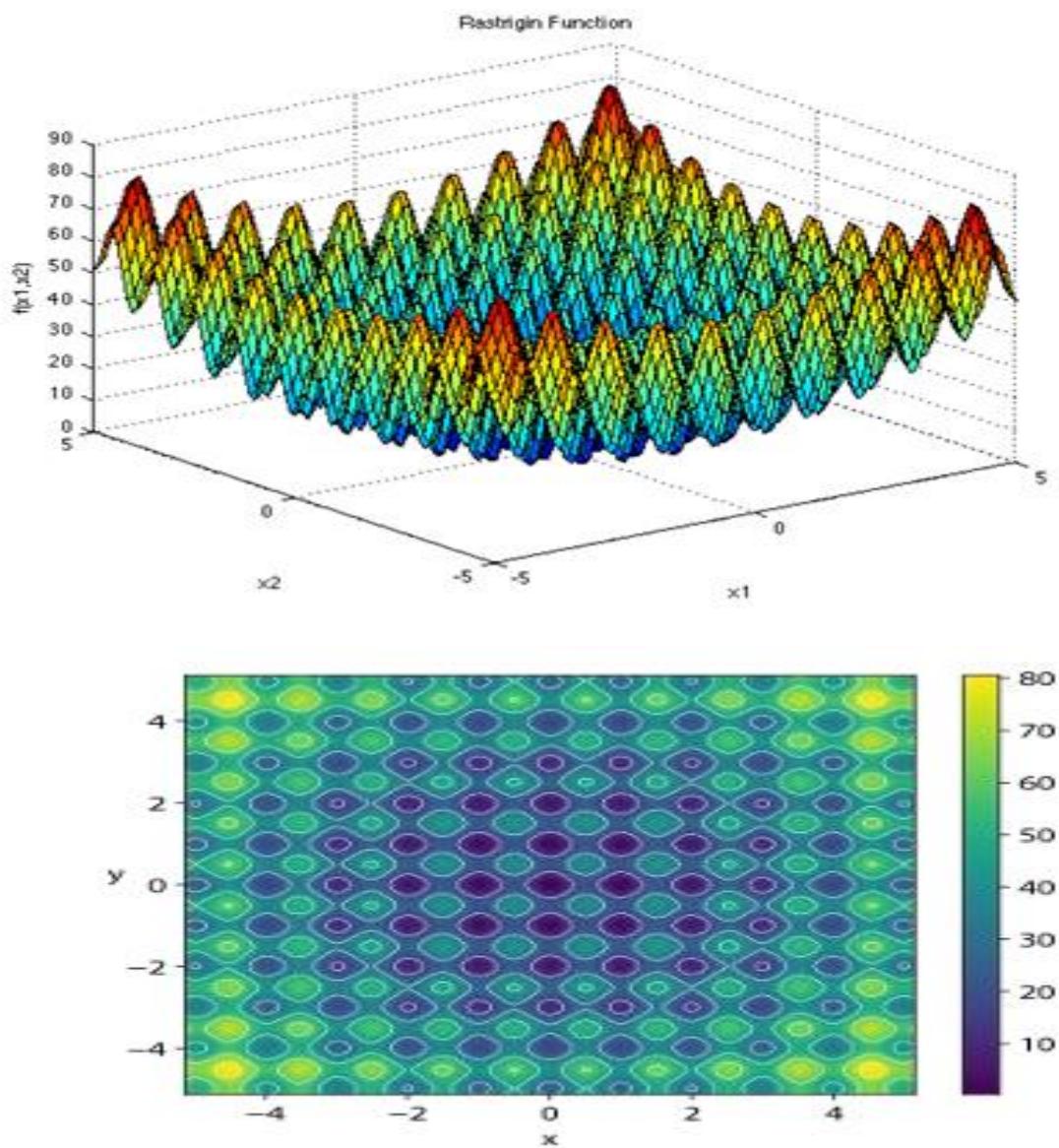


۵-بخش ۴: مسئله بهینه‌سازی تابع Rastrigin با نمایش اعشاری

تابع Rastrigin به علت داشتن تعداد زیادی کمینه محلی برای ارزیابی الگوریتم‌های تکاملی و بهینه‌سازی گزینه مناسبی است. نمودار سه بعدی این تابع در شکل ۳ نمایش داده شده است. همچنین رابطه ریاضی این تابع در زیر قابل مشاهده است:

$$f(x) = An + \sum_{i=1}^n [x_i^2 - A \cos(2\pi x_i)]$$

که در آن n تعداد بعدهای مسئله و مقدار پیشنهادی برای ضریب A برابر با 10 است.



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

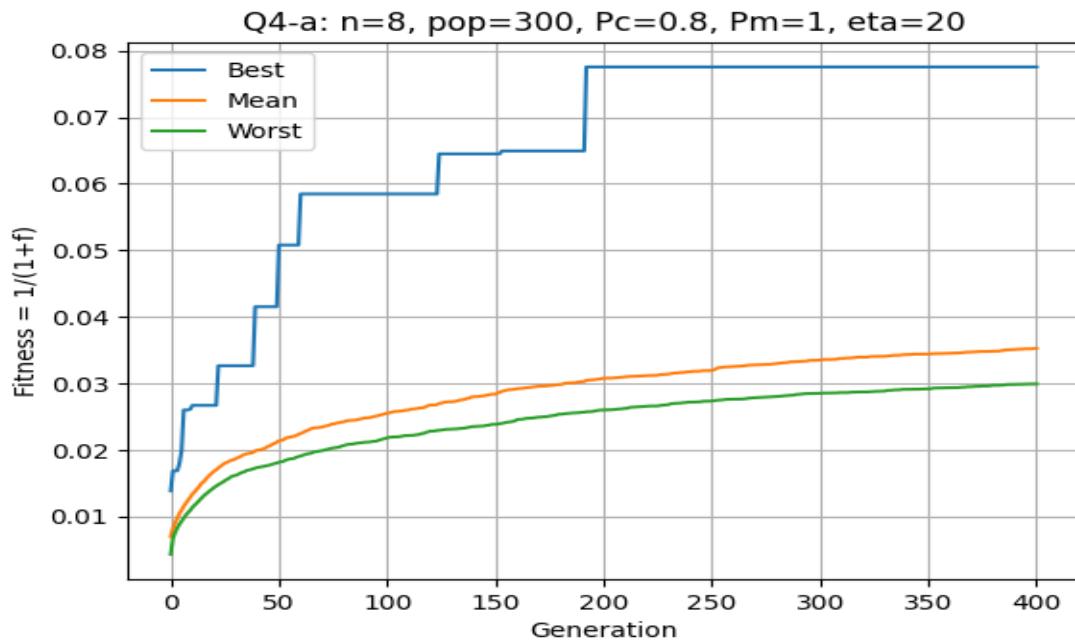
عملگرها

- 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 :
 - کوچک یعنی جهش‌های بزرگ (اکتشاف زیاد)
 - بزرگ یعنی جهش‌های کوچک یا بهره‌برداری local
- انتخاب: متناسب با برازنده‌گی (roulette) و در حالت اختیاری tournament.

الف

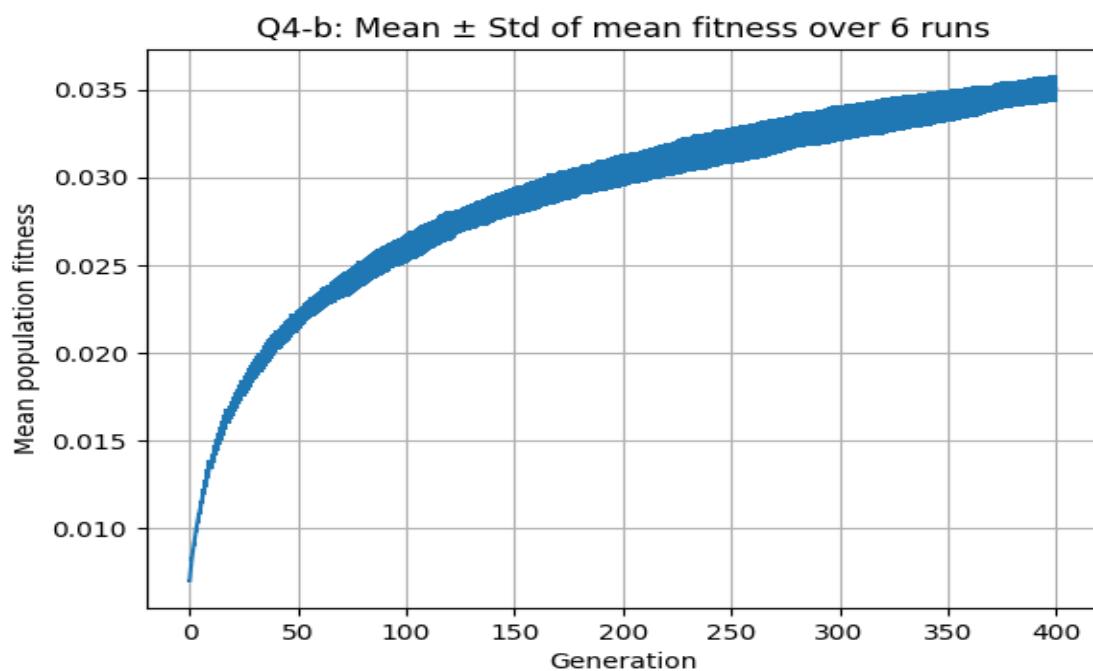
نتایج :

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



(ب)

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

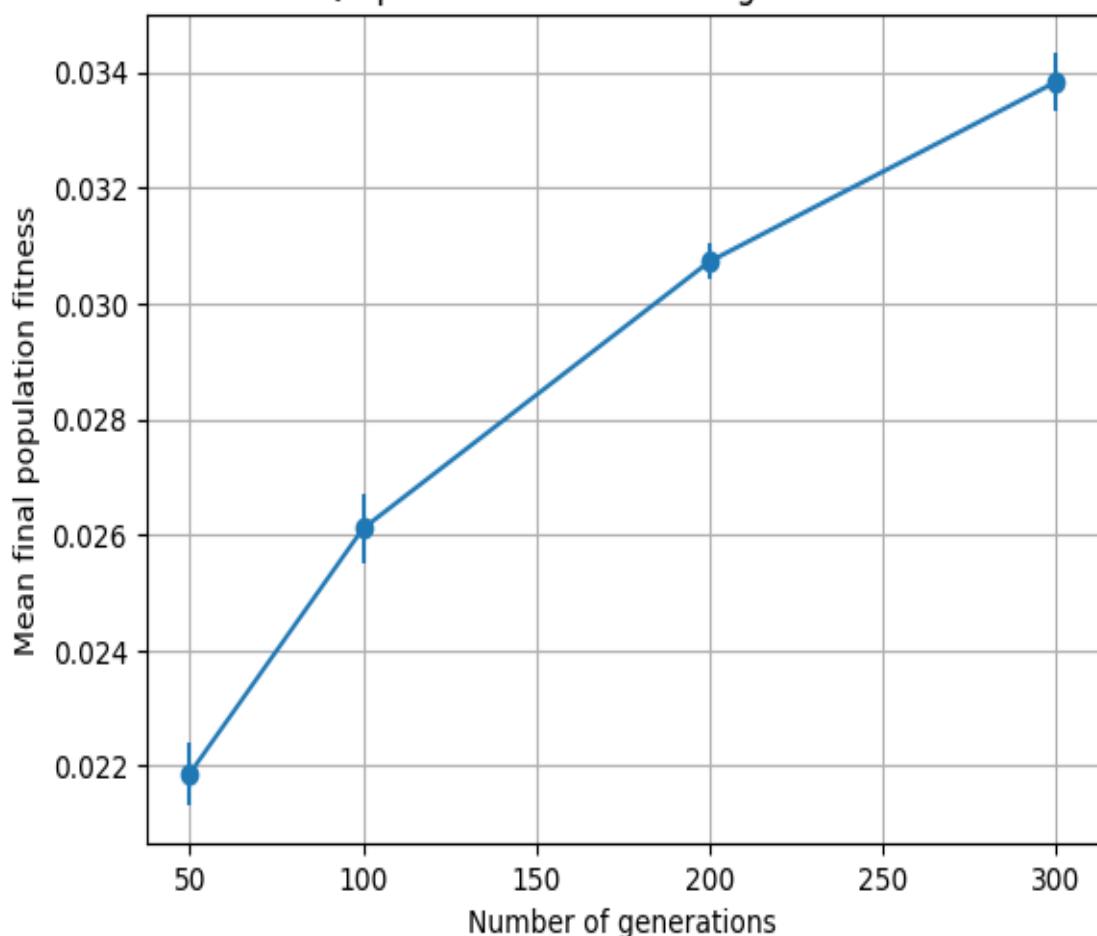


(پ)

تعداد نسل‌ها (۳۰۰، ۲۰۰، ۱۰۰، ۵۰) :

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

Q4-p: Effect of number of generations



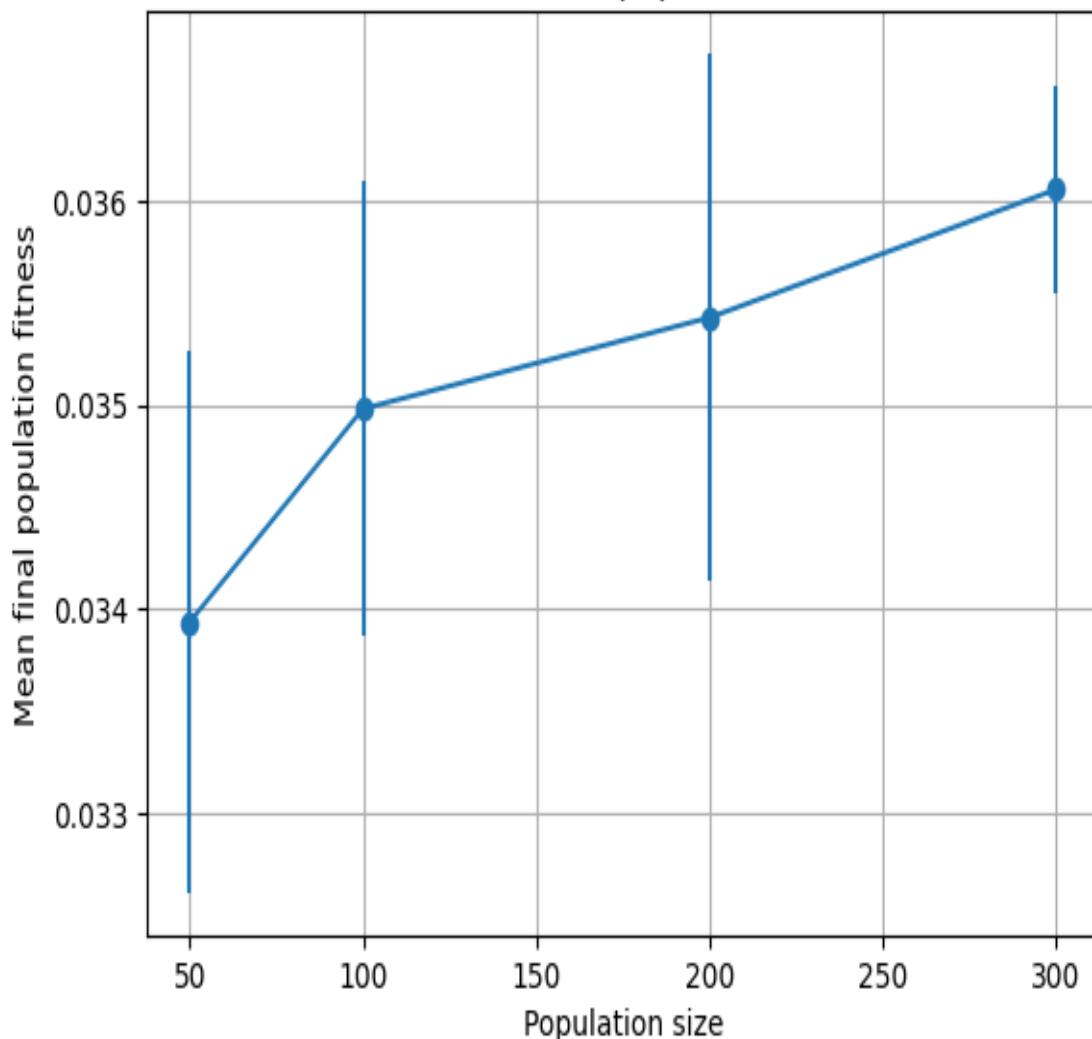
(ت)

اندازه‌ی جمعیت:

- جمعیت‌های کوچک تنوع کافی برای پوشش فضای [۵.۱۲، ۵.۱۲-۸۸] 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.

Q4-t: Effect of population size



(ث)

احتمال بازترکیب P_c :

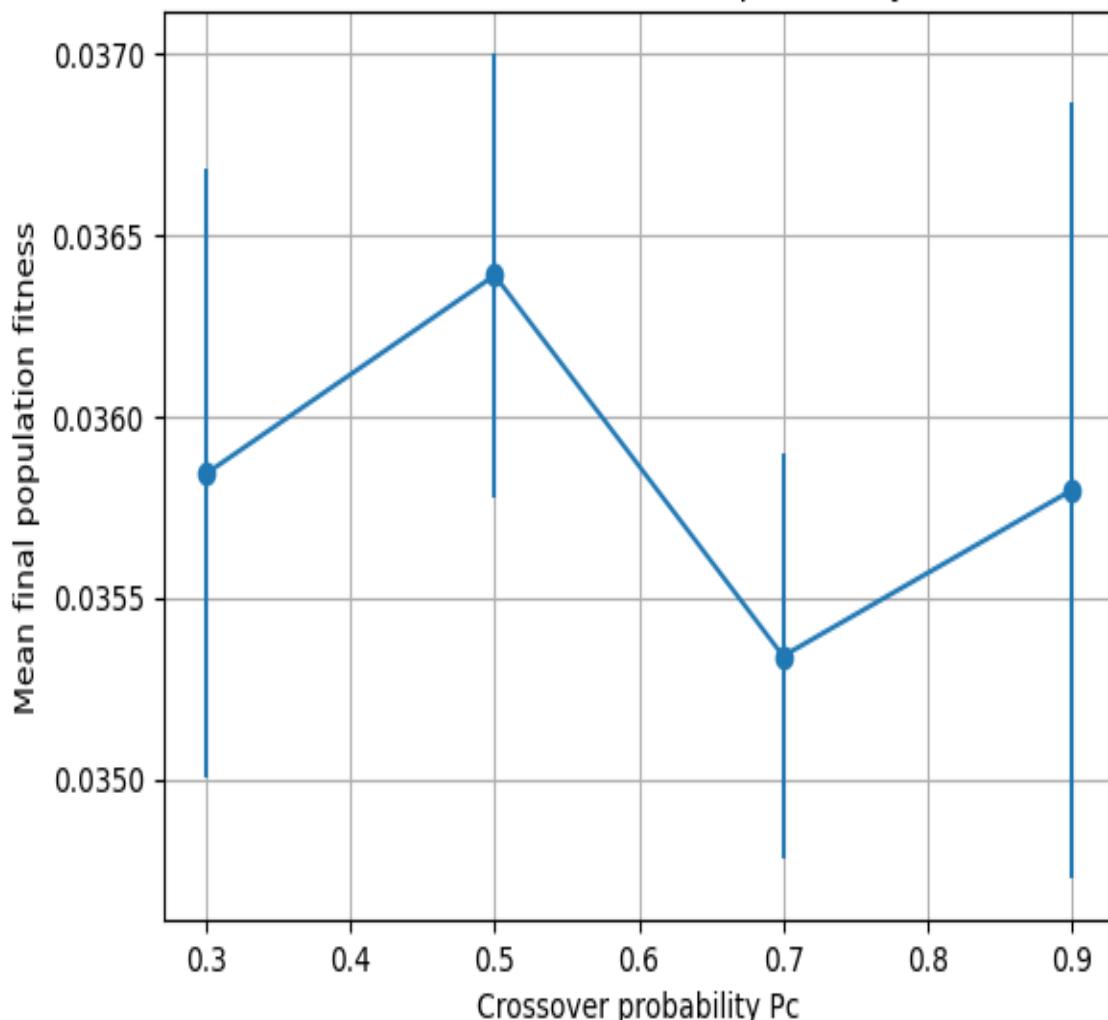
• P_c پایین یعنی offspring ها به والدین خیلی شبیه می‌مانند یا اصلاً از والدین فقط

کپی می‌شوند؛ سرعت جستجو کم می‌شود؛

• P_c بالا یعنی ترکیب قوی اطلاعات والدین، اما اگر جمعیت همگرا شده باشد، باز هم

تأثیر کمی دارد.

Q4-th: Effect of crossover probability P_c

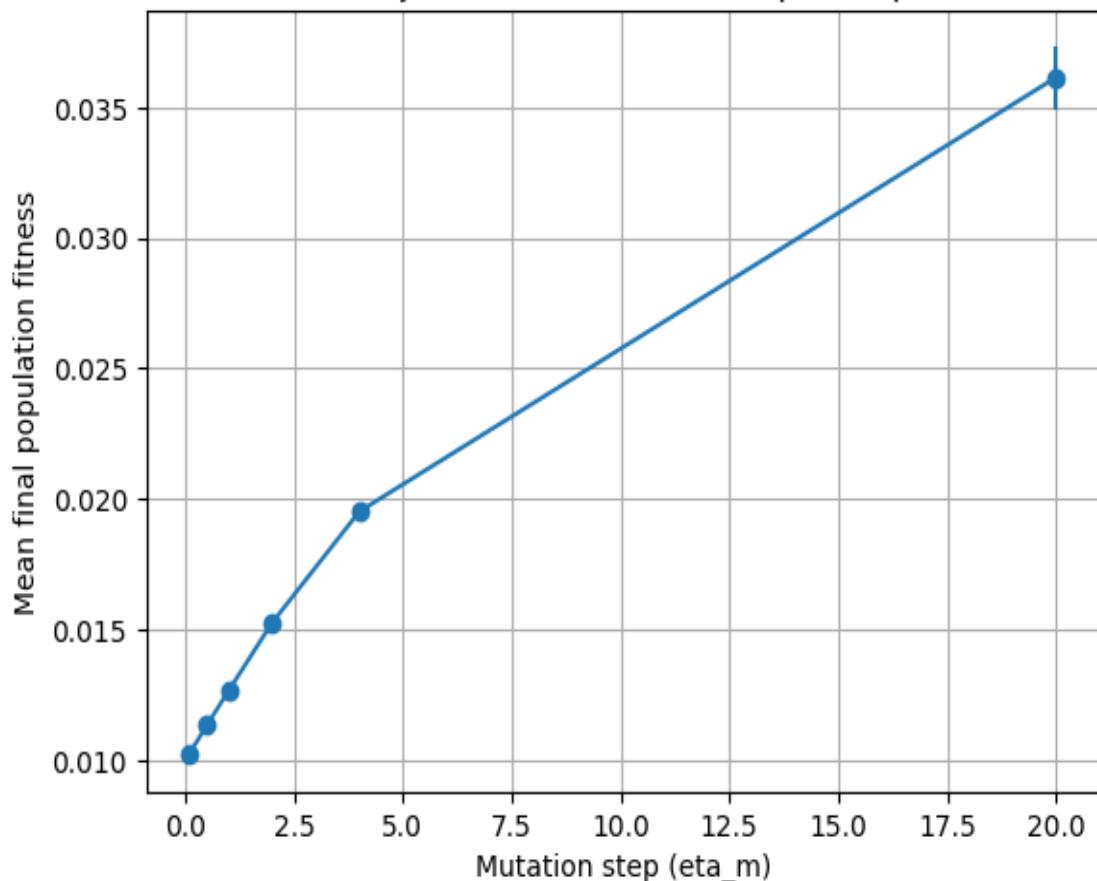


(ج)

گام جهش (η) :

- η بسیار کوچک (مثلاً ۰،۵، ۱، ۰، ۰) یعنی جهش‌های بزرگ یعنی جستجوی global قوی ولی سختی در fine-tuning نزدیک مینیمم؛
- η میانی (۰-۱) 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.

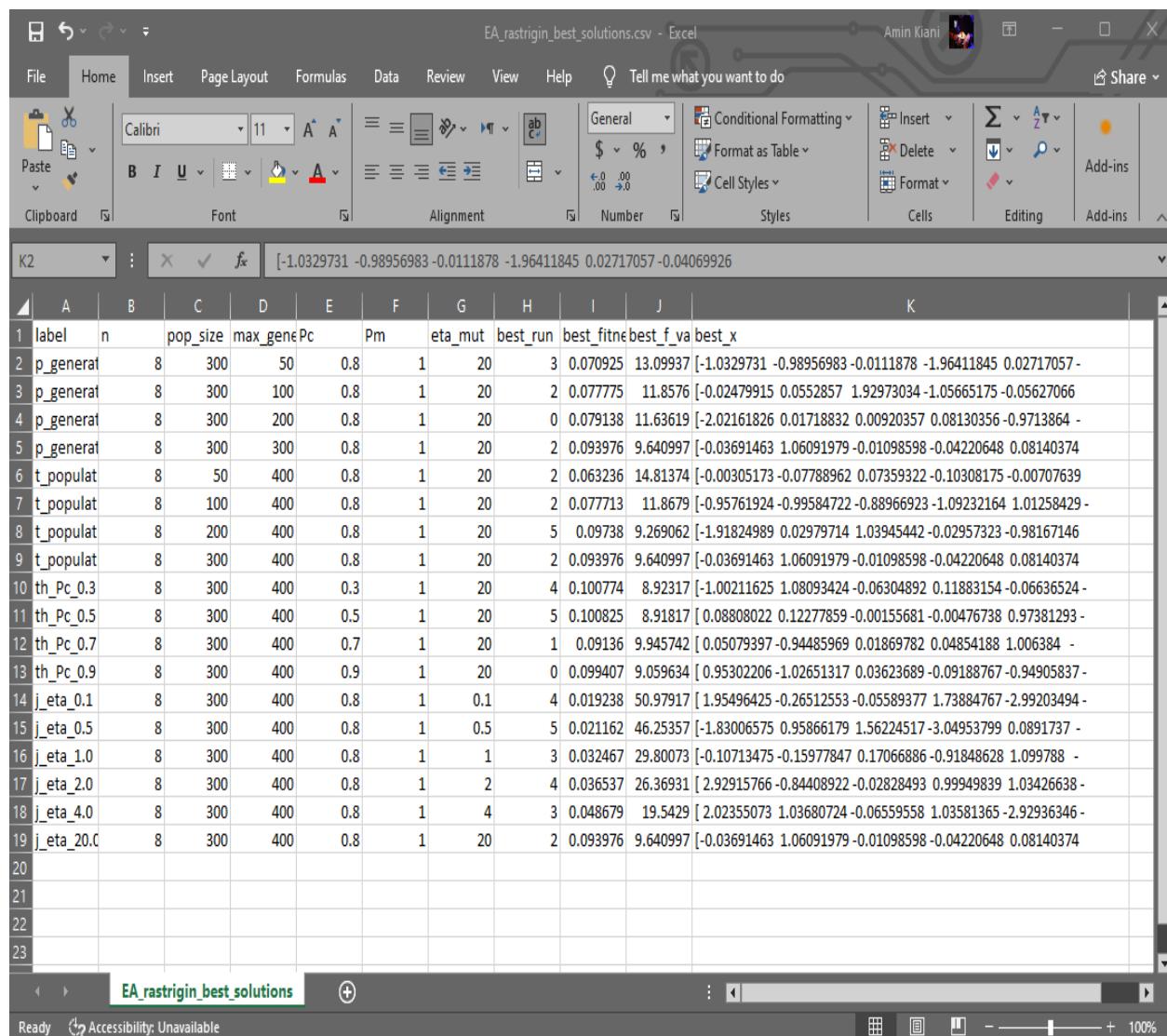
Q4-j: Effect of mutation step size η



(ج)

بهترین راه حل ها

- برای هر تنظیم، بهترین بردار x^* همراه با مقدار $f(x^*)$ در CSV ثبت شده است؛
- مقایسه تنظیم های مختلف نشان داد:
- تنظیم هایی با جمعیت بزرگ تر، PC بالا و η میانی معمولاً f پایین تری تولید می کنند؛

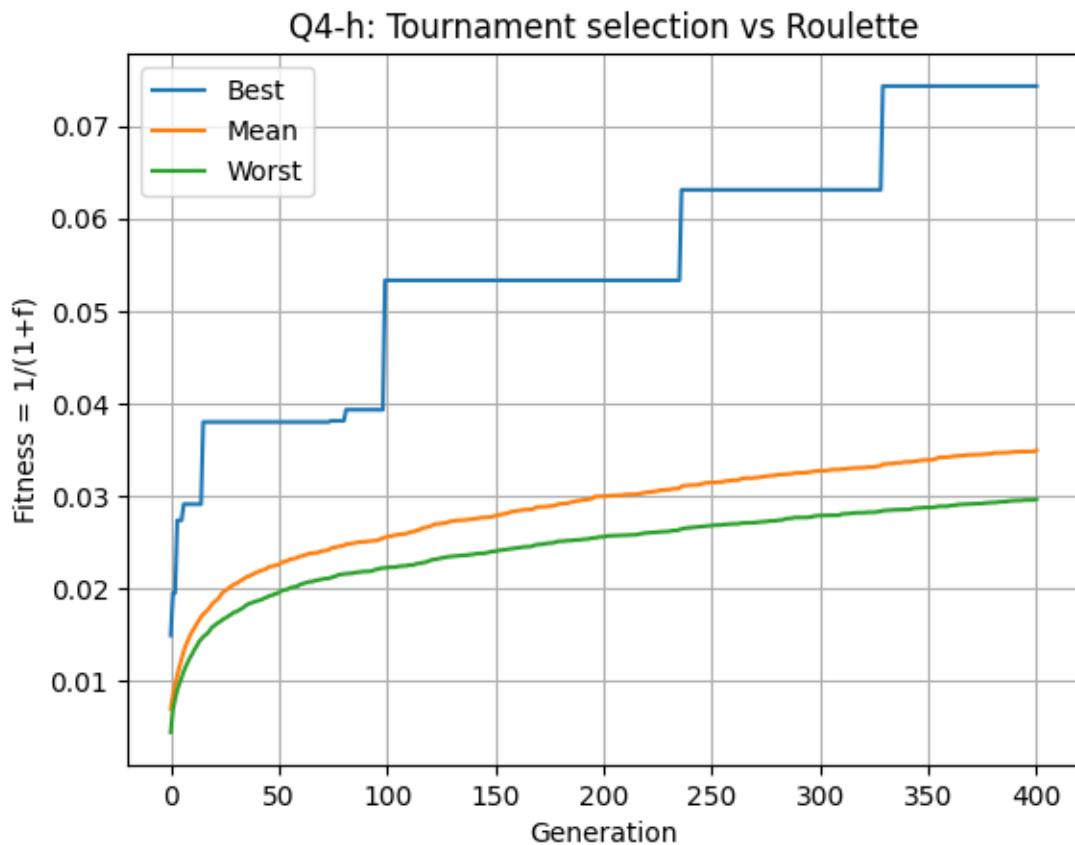


The screenshot shows an Excel spreadsheet titled 'EA_rastrigin_best_solutions.csv'. The data is organized into columns labeled A through K. Column A contains labels for parameters: 'label', 'n', 'pop_size', 'max_gen', 'Pc', 'Pm', 'eta_mut', 'best_run', 'best_fit', 'best_f_val', and 'best_x'. The subsequent rows provide data for different configurations. For example, row 2 shows results for 'p_generat' with n=8, pop_size=300, max_gen=50, and Pc=0.8. The best solution found is at index 20 with a fitness of 13.09937 and a value of -1.0329731. The last row (row 19) shows results for 'j_eta_20.0' with n=8, pop_size=300, max_gen=400, and eta_mut=0.8. The best solution found is at index 20 with a fitness of 9.640997 and a value of -0.03691463.

label	n	pop_size	max_gen	Pc	Pm	eta_mut	best_run	best_fit	best_f_val	best_x
p_generat	8	300	50	0.8	1	20	3	0.070925	13.09937	[-1.0329731 -0.98956983 -0.0111878 -1.96411845 0.02717057 -0.04069926]
p_generat	8	300	100	0.8	1	20	2	0.077775	11.8576	[-0.02479915 0.0552857 1.92973034 -1.05665175 -0.05627066]
p_generat	8	300	200	0.8	1	20	0	0.079138	11.63619	[-2.02161826 0.01718832 0.00920357 0.08130356 -0.9713864 -0.03691463]
p_generat	8	300	300	0.8	1	20	2	0.093976	9.640997	[-0.03691463 1.06091979 -0.01098598 -0.04220648 0.08140374]
t_populat	8	50	400	0.8	1	20	2	0.063236	14.81374	[-0.00305173 -0.07788962 0.07359322 -0.10308175 -0.00707639]
t_populat	8	100	400	0.8	1	20	2	0.077713	11.8679	[-0.95761924 -0.99584722 -0.88966923 -1.09232164 1.01258429 -0.03691463]
t_populat	8	200	400	0.8	1	20	5	0.09738	9.269062	[-1.91824989 0.02979714 1.03945442 -0.02957323 -0.98167146]
t_populat	8	300	400	0.8	1	20	2	0.093976	9.640997	[-0.03691463 1.06091979 -0.01098598 -0.04220648 0.08140374]
th_Pc_0.3	8	300	400	0.3	1	20	4	0.100774	8.92317	[-1.00211625 1.08093424 -0.06304892 0.11883154 -0.06636524 -0.03691463]
th_Pc_0.5	8	300	400	0.5	1	20	5	0.100825	8.91817	[0.08808022 0.12277859 -0.00155681 -0.00476738 0.97381293 -0.03691463]
th_Pc_0.7	8	300	400	0.7	1	20	1	0.09136	9.945742	[0.05079397 -0.94485969 0.01869782 0.04854188 1.006384 -0.03691463]
th_Pc_0.9	8	300	400	0.9	1	20	0	0.099407	9.059634	[0.95302206 -1.02651317 0.03623689 -0.09188767 -0.94905837 -0.03691463]
j_eta_0.1	8	300	400	0.8	1	0.1	4	0.019238	50.97917	[1.95496425 -0.26512553 -0.05589377 1.73884767 -2.99203494 -0.03691463]
j_eta_0.5	8	300	400	0.8	1	0.5	5	0.021162	46.25357	[-1.83006575 0.95866179 1.56224517 -3.04953799 0.0891737 -0.03691463]
j_eta_1.0	8	300	400	0.8	1	1	3	0.032467	29.80073	[-0.10713475 -0.15977847 0.17066886 -0.91848628 1.099788 -0.03691463]
j_eta_2.0	8	300	400	0.8	1	2	4	0.036537	26.36931	[2.92915766 -0.84408922 -0.02828493 0.99949839 1.03426638 -0.03691463]
j_eta_4.0	8	300	400	0.8	1	4	3	0.048679	19.5429	[2.02355073 1.03680724 -0.06559558 1.03581365 -2.92936346 -0.03691463]
j_eta_20.0	8	300	400	0.8	1	20	2	0.093976	9.640997	[-0.03691463 1.06091979 -0.01098598 -0.04220648 0.08140374]

(ج)

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.



٤- مراجع

- [1] <https://github.com>
- [2] <https://stackoverflow.com/questions>
- [3] <https://colab.research.google.com/>
- [4] [https://algorithmafternoon.com/books/genetic_algorithm/chapter 04/](https://algorithmafternoon.com/books/genetic_algorithm/chapter_04/)
- [5] <https://chatgpt.com/>
- [6] <https://grok.com/>