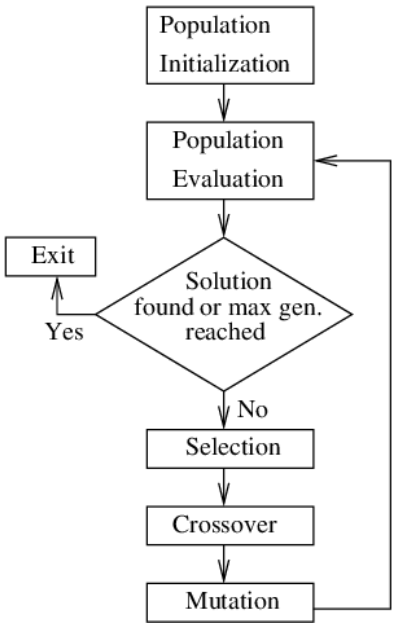
«ایجاد مصرعی از حافظ با الگوریتم تکاملی»

مهدی کافی ۹۹۲۱۰۷۵۳

در زیست شناسی، تکامل به معنای تغییر ویژگی‌های گونه‌ها در طی گذر چندین نسل و بر پایه انتخاب طبیعی است. در علم کامپیوتر نیز الگوریتم‌هایی وجود دارند که با الهام از این ویژگی طبیعت به وجود آمده‌اند و الگوریتم‌های تکاملی نامیده‌می‌شوند. الگوریتم‌های تکاملی به این صورت هستند که در ابتدا جامعه‌ای از جواب‌ها به صورت تصادفی ایجاد می‌شوند و سپس این جواب‌ها بر اساس تابعی متناسب با مسئله، امتیازدهی می‌شوند و تعدادی از نمونه‌ها که بهترین امتیاز را دارند انتخاب می‌شوند و از این تعداد به وسیله cross-over و mutation نسل بعدی ایجاد می‌شوند (تعداد افراد جامعه باید ثابت نگه داشته‌شود) و همین مراحل تکرار می‌شوند تا شرایطی پایانی دیده‌شوند که شرایط می‌توانند رسیدن به تعداد مشخصی نسل، رسیدن به امتیاز مشخص و... باشند.



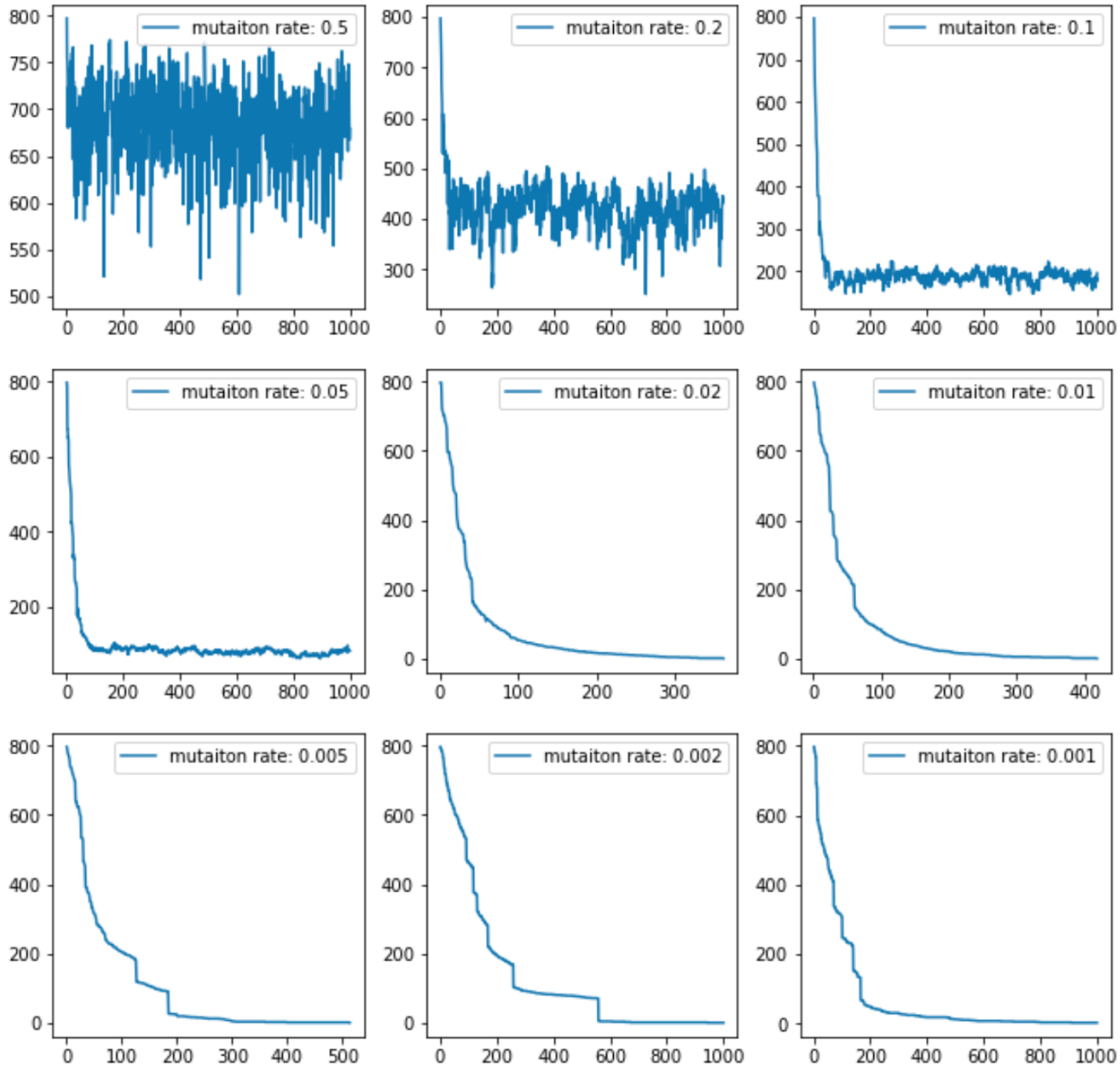
دیدگاه داروینی: اگر فرض کنیم که در ابتدا جامعه‌ای داریم که تنوع زیادی در آن‌ها وجود دارد، آن‌هایی که تناسب بهتری با محیط دارند می‌توانند زندگی بهتری داشته‌باشند و در نتیجه می‌توانند تولید مثل بیشتری داشته‌باشند و در نسل بعدی تعداد بیشتری از این نوع وجود خواهدداشت و به همین ترتیب پس از گذشت چندین نسل، تعداد افراد نوعی که بهترین تناسب را با محیط دارد نسبت به سایر انواع بسیار بیشتر خواهدبود.

دیدگاه لامارکی: اگر افراد جامعه در طول زندگی خود، ویژگی‌هایی را بر حسب نیاز در خود به وجود آورند؛ این ویژگی‌ها به نسل بعدی منتقل می‌شود.

تعریف مسأله: در این تکلیف، هدف ما ایجاد یک مصرع از شعر حافظ با استفاده از الگوریتم تکاملی است.

رویکرد حل مسأله: در این جا به صورت عمل شده‌است که در ابتدا جامعه‌ای به تعداد ۱۰۰ رشته از حروف که طول این حروف برابر با طول حروف مصرع است به صورت تصادفی ایجاد شده‌اند هر کدام از این رشته‌ها مانند کروموزوم‌ها عمل می‌کنند. سپس از این کروموزوم‌ها، ۲۵ کروموزوم برتر انتخاب می‌شوند. نحوه امتیازدهی به این صورت است که تمامی حروف رشته‌های جامعه و مصرع اصلی به کد اسکی تبدیل

می‌شوند و سپس قدر مطلق اختلاف یک به یک حروف رشته‌ها با مصرع محاسبه می‌شود و این مقادیر برای هر رشته با هم جمع می‌شوند و امتیاز آن رشته را به وجود می‌آورند بنابراین هر چه امتیاز کمتر باشد به معنای نزدیکی بیشتر رشته به مصرع اصلی و مقدار امتیاز صفر به معنای رسیدن به مصرع مورد نظر است. سپس از ۲۵ کروموزوم برتر انتخاب شده، از هر کدام ۴ فرزند با جهش ایجاد می‌شود و احتمال جهش هر حرف برابر با ۰۲/۰ در نظر گرفته شده‌است برای انتخاب احتمال جهش هر حرف، همانطور که در شکل ۱ دیده می‌شود، ۹ مقدار مختلف برای این احتمال در نظر گرفته‌شد و مشاهده‌شد که مقادیر ۰.۰۲و ۰.۰۱ با تعداد نسل کمتری به جواب میرسند.

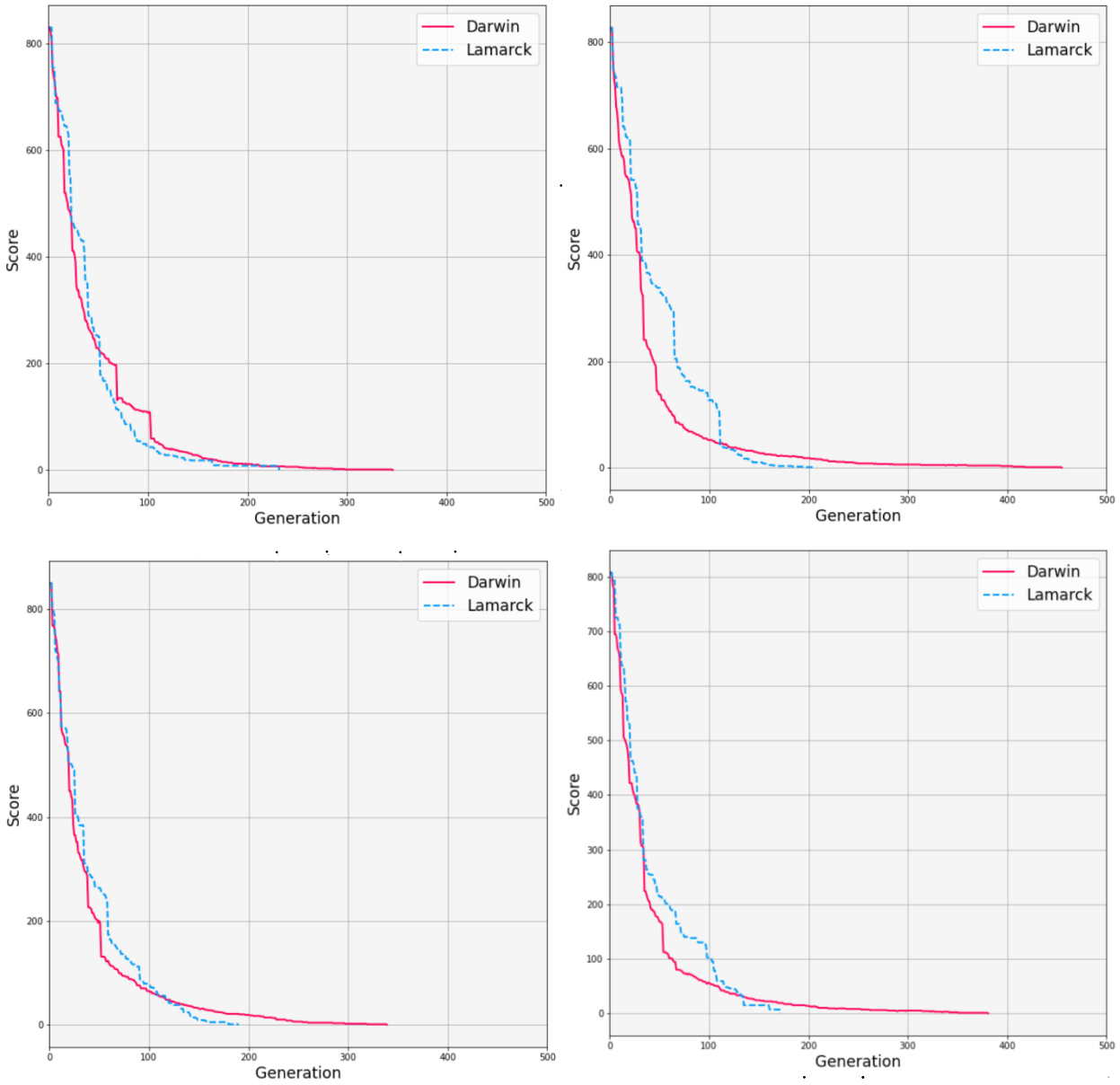


شکل ۱: ۹ مقدار مختلف برای احتمال جهش در نظر گرفته شده‌است و مقدار ۰.۰۲ به عنوان احتمال جهش انتخاب شد. مقادیر بزرگتر باعث می‌شوند که اصلا مسأله به جواب نرسد و مقادیر کوچکتر تعداد نسل بیشتری میخواهند تا به جواب برسند.

با ایجاد ۴ فرزند از هر رشته، جامعه‌ای با ۱۰۰ نفر خواهیم داشت و سپس این جامعه، والدان نسل بعدی خواهند بود و همین مراحل تکرار خواهندشد. شرایط پایانی به این صورت در نظر گرفته شده‌است که به ۱۰۰۰ نسل برسیم و یا امتیاز یکی از رشته‌ها به صفر برسد. این مراحل پیاده‌سازی دیدگاه داروین است در دیدگاه لامارک باید رشته‌ها بتوانند زندگی کنند و ویژگی‌ای را برای خود به وجود آورند، برای پیاده‌سازی

زندگی به این صورت عمل شده‌است که هر حرف هر رشته با احتمال ۰۰۰۵/۰ می‌تواند دقیقا با حرف صحیح که در مصرع آمده‌است جایگزین شود و مانند آهنگری که بازوهای بزرگی متناسب با شغل خود، کسب کرده‌است، این رشته هم خود را به مصرع نزدیکتر کرده‌است.

نمودار خروجی برنامه در ۴ بار اجرا در شکل ۲، آمده‌است.



شکل ۲: در این نمودار، محور افقی بیانگر تعداد نسل، محور عمودی برابر بهترین امتیاز هر نسل، منحنی قرمز، بیانگر دیدگاه داروینی و منحنی آبی بیانگر دیدگاه لامارکی است.

با توجه به نمودارها دیده‌می‌شود که در دیدگاه لامارکی به دلیل اینکه حروف با حروف مصرع جایگزین می‌شوند حتی با احتمال بسیار کمتر از احتمال جهش در دیدگاه داروین، با تعداد نسل کمتری، مصرع مورد نظر ساخته می‌شود. در این مسأله به دلیل اینکه فنوتایپ و ژنوتایپ در واقع یکسان و برابر حروف رشته ساخته‌شده هستند، بنابراین تفاوتی بین دیدگاه داروینی و لامارکی وجود ندارد.

کد برنامه، در ادامه آمده‌است.

```python
import string
import random
import numpy as np
import matplotlib.pyplot as plt

alphabet = string.ascii_lowercase + ' '
text = "ke eshgh asan nemud aval vali oftad moshkel ha"
print("reference text is:", text)
length = len(text)
g_1 = []
for i in range(100):
    g_1.append(random.choices(alphabet, k= len(text)))
g_1 = np.array(g_1)
ord_vec = np.vectorize(ord)
g_1 = ord_vec(g_1)
g_1_lamarck = np.copy(g_1)
reference = []
for char in text:
    reference.append(ord(char))
reference = np.array(reference)
iteration = 1000
darwin_min_scores = []
lamarck_min_scores = []
darwin_iterations = []
print("#########Darwin##########")
for it in range(iteration):
    scores = (np.absolute(g_1 - reference)).sum(axis= 1)
    # print(f"{it+1} iteration scores:", scores)
    min_score = scores.min()
    darwin_min_scores.append(min_score)
    darwin_iterations.append(it+1)
    if (it+1)%100 == 0:
        print(f"#{it+1} iteration score:", min_score)
    if min_score < 1:
        print(f"solution is found after {it+1} generation")
        min_index = scores.argmin()
        solution = g_1[min_index]
        # print(solution)
        vec_chr = np.vectorize(chr)
        hafez = vec_chr(solution)
        # print(hafez)
        print("solution found by Darwin hypothesis:", ''.join(hafez))
        break
    k = 25
    best_g_1_mask = np.zeros(g_1.shape[0], dtype=np.bool)
```

```
    for i in range(k):
        argmin = scores.argmin()
        best_g_1_mask[argmin] = 1
        scores[argmin] = 10000
    # print(best_g_1_mask)
    g_1_best = g_1[best_g_1_mask]
    offsprings = np.empty((1, length), dtype= np.int16)
    ###mutation###
    mutation_list = list(range(97, 123))
    mutation_list.append(32)
    mutation_rate = 0.02
    for i in range(g_1_best.shape[0]):
        for k in range(4):
            child = np.copy(g_1_best[i])
            for j in range(g_1_best.shape[1]):
                if random.random() < mutation_rate:
                    child[j] = random.choice(mutation_list)
            offsprings = np.concatenate((offsprings, [child]), axis= 0)
    offsprings = np.delete(offsprings, 0, axis=0)
    # print(offsprings)
    g_1 = offsprings

"""Lamarck Hypothesis"""

g_1 = g_1_lamarck
iteration = 1000
lamarck_min_scores = []
lamarck_iterations = []
print("#########Lamarck#########")
for it in range(iteration):
    scores = (np.absolute(g_1 - reference)).sum(axis= 1)
    # print(f"{it+1} iteration scores:", scores)
    min_score = scores.min()
    lamarck_min_scores.append(min_score)
    lamarck_iterations.append(it+1)
    if (it+1)%100 == 0:
        print(f"#{it+1} iteration score:", min_score)
    if min_score < 1:
        print(f"solution is found after {it+1} generation")
        min_index = scores.argmin()
        solution = g_1[min_index]
        # print(solution)
        vec_chr = np.vectorize(chr)
        hafez = vec_chr(solution)
        # print(hafez)
```

```
        print("solution found by Lamarck hypothesis:", ''.join(hafez))
        break
    k = 25
    best_g_1_mask = np.zeros(g_1.shape[0], dtype=np.bool)
    for i in range(k):
        argmin = scores.argmin()
        best_g_1_mask[argmin] = 1
        scores[argmin] = 10000
    # print(best_g_1_mask)
    g_1_best = g_1[best_g_1_mask]
    offsprings = np.empty((1, length), dtype= np.int16)
    ###mutation###
    learning_rate = 0.0005
    for i in range(g_1_best.shape[0]):
        for k in range(4):
            child = np.copy(g_1_best[i])
            for j in range(g_1_best.shape[1]):
                if random.random() < learning_rate:
                    child[j] = reference[j]
            offsprings = np.concatenate((offsprings, [child]), axis= 0)
    offsprings = np.delete(offsprings, 0, axis=0)
    # print(offsprings)
    g_1 = offsprings
fig, ax= plt.subplots(figsize=(10, 10))
ax.plot(darwin_iterations, darwin_min_scores, label="Darwin", color="#FF005E", li
newidth=2)
ax.plot(lamarck_iterations, lamarck_min_scores, color= "#0099FF", label="Lamarck"
, linestyle="--", lw= 2)
ax.set_xlim(0, 500)
ax.grid(True)
ax.legend(fontsize="xx-large")
ax.set_facecolor('#f5f5f5')
ax.set_ylabel("Score", fontsize="xx-large")
ax.set_xlabel("Generation", fontsize="xx-large")
```