دانشکده مهندسی برق

گزارش پروژه طراحی در سطح سیستم

عنوان پروژه:

Evolutionary Strategy for Embedded Systems

نام و نام خانوادگی دانشجویان:

علی اکبر محسن نژاد – ۴۰۰۰۸۸۷۳

پارسا رشتیان _ ۴۰۰۱۰۸۲۳

نام و نام خانوادگی استاد درس:

دکتر هدی رودکی لواسانی

نام و نام خانوادگی دستیار آموزشی مرتبطه:

سرکارخانم اطهر حکیم‌زاده

تاریخ انجام پروژه:

۱۴۰۴/۰۳/۲۰ لغایت ۱۴۰۴/۰۵/۰۶

فهرست مطالب

۱- مقدمه

در مسالهای بهینه‌سازی که با قیود و محدودیت‌های معین مواجه هستیم، مسئله کوله‌پشتی (Knapsack Problem) از مهم‌ترین مسائل مرجع در رشته‌های مهندسی و علوم محاسباتی است. هدف اصلی انتخاب ترکیبی از ایتم‌هاست که با رعایت محدودیت وزنی، بیشترین ارزش (value) ممکن را تولید کند.

در پروژه حاضر با هدف بهینه‌سازی این مسئله در ساختار هم‌طراحی سخت‌افزار/نرم‌افزار(HW/SW Co-Design) ، از الگوریتم استراتژی تکاملی (Evolution Strategy) استفاده شده است. با تقسیم موردی منطقی بین ماژول‌های سخت‌افزاری و نرم‌افزاری، عملیات بازترکیب و جهش در بخش SystemC و بررسی شایستگی و انتخاب گزینه در MATLAB انجام شده است.

هفت مولفه کلیدی در این پروژه در نظر گرفته شد:

- مدل مسئله‌ای دقیق
- انتخاب الگوریتم مناسب
- شبیه‌سازی بخش نرم‌افزار
- شبیه‌سازی بخش سخت‌افزار
- برقراری ارتباط دقیق بین آنها
- انتخاب سازوکار توقف معتبر
- و ساختار ماژولار در هر دو بخش

در ادامه، هر کدام از این موارد تفصیلاً توضیح داده می‌شود.

۲-توصیف مسئله و مدل

۲-۱ نوع مسئله و فرضیات

در این پروژه، نسخه پیوسته‌ی مسئله کلاسیک کوله‌پشتی مورد استفاده قرار گرفته است. برخلاف مدل دودویی (که در آن هر آیتم یا انتخاب می‌شود یا نمی‌شود)، در اینجا می‌توان درصدی از هر آیتم را انتخاب کرد. به‌عبارت دیگر، متغیرهای انتخاب‌شده می‌توانند مقداری بین ۰ و ۱ داشته باشند. این مدل در کاربردهایی که انتخاب نسبی معنا دارد (مثلاً تخصیص منابع به‌صورت جزئی)، پرکاربرد است.

۲-۲ پارامترها و متغیرهای مدل

برای هر آیتم، دو پارامتر اصلی تعریف شده است:

value(ارزش): سود یا اهمیت انتخاب آیتم

weight(وزن): میزان منابع مصرفی یا محدودیت مرتبط با آیتم

متغیرهای تصمیم مدل به‌صورت بردار $x_i = [x_1, x_2, \dots, x_8]$ هستند که هر x_i درصد انتخاب آیتم i ام را نشان می‌دهد.

۲-۳ تابع هدف و محدودیت

هدف، بیشینه‌سازی مجموع ارزش انتخاب‌شده‌ها است:

Maximize: $\sum x_i \,.\, value_i$

با این محدودیت که مجموع وزن نباید از حد مجاز فراتر رود:

$$\sum x_i \,.\, weight_i \;\leq \max - \text{weight}$$

اگر وزن کل ترکیب پیشنهادی بیش از حد مجاز باشد، شایستگی آن با یک **تابع جریمه خطی** کاهش داده می‌شود.

۴-۲ داده‌های ورودی استفاده‌شده

- تعداد آیتم‌ها: ۸
- بردار ارزش: [6, 5, 8, 9, 6, 7, 3, 6]
- بردار وزن: [2, 3, 6, 7, 5, 9, 3, 4]
- حداکثر وزن مجاز: ۱۵ واحد

۵-۲ هدف در هم‌طراحی سخت‌افزار/نرم‌افزار

در این پروژه، هدف از هم‌طراحی این بوده که وظایف محاسباتی پرهزینه مانند عملیات بازترکیب و جهش در بخش سخت‌افزار (SystemC) اجرا شوند، درحالی‌که بخش‌هایی مانند انتخاب والدین، ارزیابی شایستگی و شرط توقف در MATLABباقی بمانند. این تقسیم باعث تسریع روند اجرا و بهره‌وری بیشتر از منابع سخت‌افزاری و نرم‌افزاری شده است.

۳-الگوریتم استراتژی تکاملی

الگوریتم‌های تکاملی از جمله روش‌های فراابتکاری هستند که با الهام از فرآیندهای طبیعی مانند انتخاب طبیعی و جهش ژنتیکی، مسائل بهینه‌سازی را حل می‌کنند. در این پروژه، الگوریتم استراتژی تکاملی (Evolution Strategy) به کار گرفته شده است. ساختار این الگوریتم به صورت زیر بوده و در دو بخش نرم‌افزار و سخت‌افزار تقسیم شده است.

۱-۳ مراحل کلی الگوریتم

۱-۱-۳ تولید جمعیت اولیه

بردارهایی با مقادیر تصادفی در بازه [۰,۱] برای هر آیتم تولید می‌شوند.
هر بردار نمایانگر یک راه‌حل پیشنهادی برای ترکیب انتخاب آیتم‌ها است.

۲-۱-۳ ارزیابی شایستگی(Fitness Evaluation)

مجموع ارزش آیتم‌های انتخاب‌شده (به نسبت درصد انتخاب‌شده) محاسبه می‌شود.
اگر وزن کل راه‌حل بیش از حد مجاز باشد، جریمه روی مقدار شایستگی اعمال می‌شود.

۳-۱-۳ انتخاب والدین(Parent Selection)

روش تورنمنت برای انتخاب والدین استفاده می‌شود. در هر تورنمنت چند عضو تصادفی مقایسه می‌شوند و بهترین به‌عنوان والد انتخاب می‌گردد.

۳-۱-٤ بازترکیب و جهش(Crossover & Mutation)

عملیات بازترکیب و جهش توسط بخش سخت‌افزاری در SystemC انجام می‌شود.
بازترکیب به‌صورت تک‌نقطه‌ای (one-point crossover) انجام شده و جهش با نویز گاوسی روی ژن‌ها اعمال می‌شود.

۳-۱-٥ ارزیابی فرزندان(Offspring Evaluation)

مشابه مرحله ۲، فرزندان ارزیابی می‌شوند.

۳-۱-٦ انتخاب جمعیت جدید(Survivor Selection)

ترکیب جمعیت فعلی و فرزندان به‌صورت یکپارچه ارزیابی شده و بهترین‌ها برای نسل بعد انتخاب می‌شوند.

۳-۱-۷ شرط توقف

اگر پس از حداقل ۵ نسل، بهترین مقدار شایستگی در ۵ نسل متوالی بدون تغییر باقی بماند، الگوریتم متوقف می‌شود.

۳-۲ تفکیک سخت‌افزار و نرم‌افزار

بخش	وظایف انجام شده
نرم‌افزار (MATLAB)	تولید جمعیت، انتخاب والدین، ارزیابی شایستگی، نمایش خروجی، شرط توقف
سخت‌افزار (SystemC)	انجام بازترکیب و جهش به صورت موازی و پرسرعت

ارتباط بین MATLAB و SystemC با استفاده از فایل‌های متنی موقت انجام می‌شود که والدین و پارامترها از طریق فایل‌ها به SystemC ارسال شده و خروجی فرزندان دریافت می‌شود.

۴-معماری سیستم هم‌طراحی

در این بخش، نحوه سازمان‌دهی و تعامل بین اجزای سخت‌افزاری و نرم‌افزاری پروژه بررسی می‌شود. طراحی پروژه به گونه‌ای انجام شده است که با تفکیک وظایف و ارتباط فایل‌محور بین دو بخش، ساختار هم‌طراحی حفظ شود.

۱-۴ معماری نرم‌افزاری(MATLAB)

بخش نرم‌افزاری پروژه در محیط MATLAB پیاده‌سازی شده است و وظیفه مدیریت و اجرای حلقه اصلی الگوریتم تکاملی را بر عهده دارد. ساختار نرم‌افزاری به صورت ماژولار طراحی شده و به‌صورت تابعی تقسیم‌بندی شده است. وظایف اصلی MATLABبه شرح زیر هستند:

٤-۱-۱ تولید جمعیت اولیه

در ابتدا، جمعیتی از راه‌حل‌های اولیه به‌صورت تصادفی ایجاد می‌شود. هر راه‌حل برداری با مقادیر پیوسته در بازه [۰, ۱] برای هر آیتم است.

٤-۱-۲ ارزیابی شایستگی

با استفاده از تابع evaluate_fitness، برای هر کروموزوم (بردار راه‌حل)، مقدار شایستگی محاسبه می‌شود. در صورت تجاوز از محدودیت وزن، مقدار شایستگی با تابع جریمه کاهش می‌یابد.

٤-۱-۳ انتخاب والدین

والدین برای تولید نسل بعد با روش تورنمنت انتخاب می‌شوند، که در آن چندین فرد به‌صورت تصادفی انتخاب و بهترین آن‌ها انتخاب می‌شود.

٤-۱-٤ ارسال داده به سخت‌افزار

والدین انتخاب‌شده و پارامترهای جهش و بازترکیب در قالب فایل‌های متنی (مانند matlab_to_systemc.dat و params.dat) ذخیره می‌شوند تا توسط SystemC پردازش شوند.

٤-۱-٥ دریافت خروجی از سخت‌افزار

نتایج حاصل از سخت‌افزار در فایل systemc_to_matlab.dat خوانده می‌شود. سپس فرزندان تولیدشده به جمعیت افزوده می‌شوند.

٤-۱-٦ انتخاب نسل بعد

با ترکیب جمعیت والد و فرزندان، بهترین افراد برای نسل بعد انتخاب می‌شوند.

٤-۱-۷ شرط توقف

الگوریتم زمانی متوقف می‌شود که مقدار شایستگی به مدت ۵ نسل متوالی (پس از حداقل ۵ نسل اول) بدون تغییر باقی بماند.

۴-۱-۸ نمایش نتایج و رسم نمودار

در پایان اجرای الگوریتم، بهترین راه‌حل به همراه وزن، ارزش، درصد استفاده از ظرفیت، و نمودار همگرایی نمایش داده می‌شود.

۴-۱-۹ ساختار ماژولار فایل‌ها

فایل‌های کلیدی MATLAB شامل موارد زیر هستند:

- main_script.m (مدیریت کلی اجرای الگوریتم)
- evolutionary_strategy_knapsack.m (حلقه اصلی تکاملی)
- evaluate_fitness.m (ارزیابی شایستگی)
- select_parents.m (انتخاب والدین)
- hardware_accelerated_operations.m (تعامل با سخت‌افزار)

۴-۲ معماری سخت‌افزاری(SystemC)

بخش سخت‌افزاری پروژه با استفاده از زبان SystemC پیاده‌سازی شده است. هدف از این بخش، اجرای سریع و مستقل عملیات‌های تکاملی پرمحاسبه مانند بازترکیب (Crossover) و جهش (Mutation) در محیط شبیه‌سازی‌شده سخت‌افزاری است. این ساختار به کاهش بار پردازشی MATLAB و افزایش هم‌زمانی و سرعت الگوریتم کمک می‌کند.

۴-۲-۱ ساختار فایل‌های SystemC

معماری سخت‌افزاری از چند فایل ماژولار تشکیل شده است:

- main.cpp — کنترل ورودی/خروجی و اجرای ماژول
- evolutionary_strategy.h — تعریف ماژول اصلی
- mutation.h — ماژول مستقل برای جهش
- recombination.h — ماژول مستقل برای بازترکیب

۴-۲-۲ ماژول اصلی EvolutionStrategy

این ماژول هسته مرکزی سخت‌افزار محسوب می‌شود و وظیفه هماهنگ‌سازی دو عملگر بازترکیب و جهش را بر عهده دارد. ساختار آن شامل موارد زیر است:

- تعریف پارامترهای الگوریتم (مانند نرخ جهش و بازترکیب)
- حلقه پردازش والدین در تابع (process_parents)
- تخصیص به ماژول‌های فرعی recombine و mutate

٤-٢-٣ عملگر بازترکیب(recombination.h)

در این بخش، با استفاده از بازترکیب تک‌نقطه‌ای، دو والد وارد شده و دو فرزند جدید تولید می‌شوند. محل کراس‌اور به‌صورت تصادفی انتخاب می‌شود.

٤-٢-٤ عملگر جهش(mutation.h)

این ماژول بر اساس احتمال جهش تعیین‌شده، مقدار نویز گاوسی به ژن‌های منتخب اضافه می‌کند. مقادیر اصلاح‌شده در محدوده [۰, ۱] نگه‌داری می‌شوند.

٤-٢-٥ مدیریت ورودی و خروجی فایل‌ها

داده‌های ورودی از فایل‌های matlab_to_systemc.dat و params.dat خوانده می‌شوند.
فرزندان تولیدشده در فایل systemc_to_matlab.dat نوشته می‌شوند.
تمامی عملیات به صورت دوره‌ای بررسی و اجرا می‌شوند (با وقفه زمانی ۱۰۰ ms) در main.cpp.

٤-٣ روش ارتباط بین MATLAB و SystemC

در این پروژه، تبادل داده بین بخش نرم‌افزاری (MATLAB) و بخش سخت‌افزاری (SystemC) از طریق فایل‌های متنی مشترک با فرمت (dat.) انجام می‌شود. این روش ساده، قابل فهم و مناسب برای شبیه‌سازی آفلاین است و نیاز به پیاده‌سازی پروتکل‌های پیچیده ارتباطی را حذف می‌کند.

٤-٣-١ فرآیند ارسال داده از MATLAB به SystemC

MATLAB پس از انتخاب والدین و تعیین پارامترهای الگوریتم، داده‌های زیر را در فایل‌ها ذخیره می‌کند:

- matlab_to_systemc.dat → حاوی ماتریس والدین (هر ردیف یک والد)
- params.dat → شامل نرخ جهش و نرخ بازترکیب به‌صورت دو مقدار اعشاری

این فایل‌ها توسط SystemC خوانده می‌شوند تا عملیات تکاملی آغاز شود.

٤-٣-٢ خواندن و پردازش داده‌ها در SystemC

در SystemC، در فایل main.cpp یک حلقه بی‌نهایت وجود دارد که به‌صورت دوره‌ای (هر ۱۰۰ms) بررسی می‌کند آیا فایل matlab_to_systemc.dat ایجاد شده است یا خیر. اگر فایل وجود داشته باشد:

- مقادیر فایل‌ها خوانده می‌شود.
- عملیات بازترکیب و جهش توسط ماژول EvolutionaryStrategy اجرا می‌شود.
- نتایج در systemc_to_matlab.dat ذخیره می‌شوند.

٤-٣-٣ دریافت نتایج توسط MATLAB

پس از اجرای SystemC و تولید فایل خروجی :systemc_to_matlab.dat

MATLAB تا زمان وجود این فایل صبر می‌کند.
پس از مشاهده فایل، آن را می‌خواند و داده‌های فرزندان را دریافت می‌کند.
در پایان، فایل حذف می‌شود تا از تکرار خواندن جلوگیری شود.

۵-کد نویسی و توضیحات

۵-۱ بخش نرم افزاری

۵-۱-۱ فایل main_script.m

این اسکریپت، نقطه ورود اصلی برای اجرای پروژه هم‌طراحی سخت‌افزار و نرم‌افزار است و وظیفه مدیریت اجرای الگوریتم، تعامل با بخش سخت‌افزاری، تحلیل خروجی‌ها و رسم نمودارها را دارد.
کد کامل main_script.m با شرح عملکرد:

```
% main_script.m
clear; close all; clc;

% نمایش عنوان
fprintf('=== پروژه هم‌طراحی سخت‌افزار و نرم‌افزار ===\n');
fprintf('پیاده‌سازی الگوریتم استراتژی‌های تکاملی برای مسئله کوله‌پشتی\n');

% تنظیمات اولیه
use_hardware = false;  % true = با SystemC، false = فقط MATLAB
```

انتخاب حالت اجرای پروژه: اگر true باشد، الگوریتم با SystemC تعامل خواهد داشت.

```
% پاکسازی فایل‌های قبلی
delete_if_exists('matlab_to_systemc.dat');
delete_if_exists('systemc_to_matlab.dat');
delete_if_exists('params.dat');
```

حذف فایل‌های متنی قدیمی برای جلوگیری از تداخل بین اجراها.

```
% اجرای الگوریتم
tic;
if use_hardware
    [best_solution, best_fitness, history] = evolutionary_strategy_knapsack();
else
    [best_solution, best_fitness, history] = evolutionary_strategy_knapsack_software();
end
time_elapsed = toc;
```

اجرای الگوریتم با توجه به حالت انتخاب‌شده و محاسبه زمان اجرا.

```
% محاسبه خروجی‌ها
items_price = [6, 5, 8, 9, 6, 7, 3, 6];
items_weight = [2, 3, 6, 7, 5, 9, 3, 4];
max_weight = 15;
total_weight = sum(best_solution .* items_weight);
total_value = sum(best_solution .* items_price);
selected_items = sum(best_solution > 0.5);  % تعداد آیتم‌های با انتخاب بالا
```

محاسبه وزن، ارزش کل و تعداد آیتم‌های انتخاب‌شده از ترکیب بهینه.

```
% فقط یک بار نمایش نتایج
fprintf('=== نتایج نهایی ===\n');
fprintf('بهترین ترکیب یافت شده (درصد انتخاب هر آیتم):\n');
disp(best_solution);

fprintf('ارزش: %.2f | وزن کل: %d / %d | شایستگی: %.2f\n', ...
    best_fitness, total_weight, max_weight, total_value);
fprintf('آیتم‌های انتخاب‌شده: %d از %d | ظرفیت: %.2f%% استفاده\n', ...
    selected_items, length(best_solution), (total_weight / max_weight) * 100);
fprintf('زمان اجرا: %.2f ثانیه\n', time_elapsed);
```

چاپ نتایج نهایی شامل ترکیب بهینه، شایستگی، استفاده از ظرفیت و زمان اجرا.

```
% حذف فایل‌های موقت در پایان
delete_if_exists('matlab_to_systemc.dat');
delete_if_exists('systemc_to_matlab.dat');
delete_if_exists('params.dat');
```

پاکسازی فایل‌های موقت خروجی و ورودی برای اجرای تمیز بعدی.

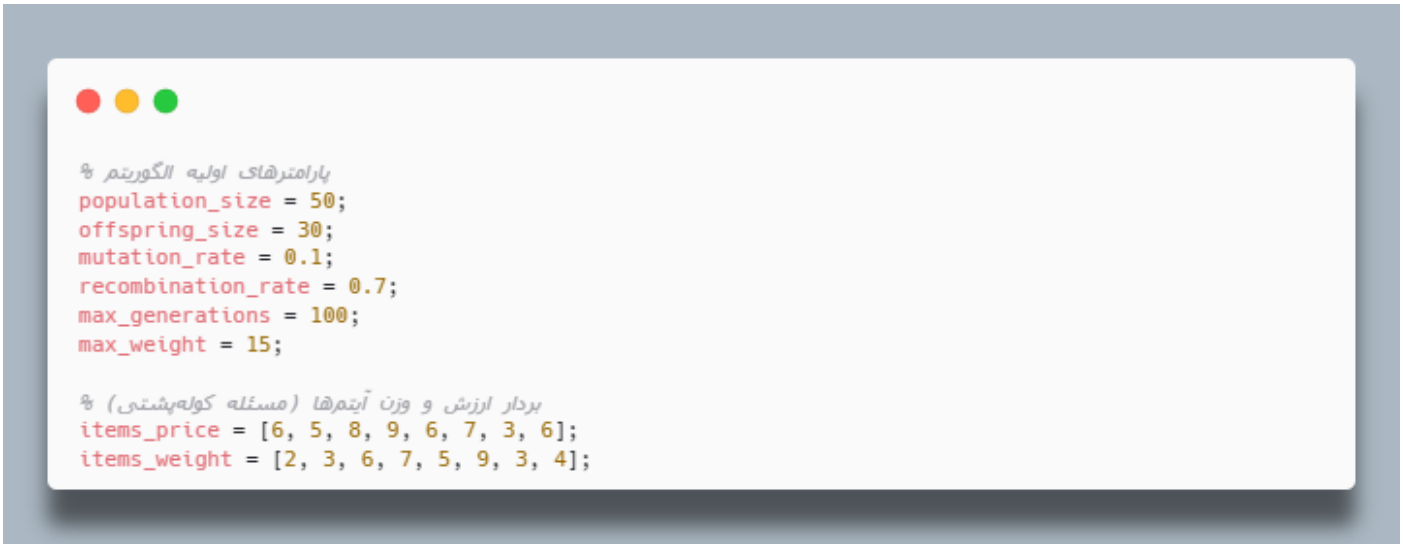
```
% رسم نمودار همگرایی
valid_length = find(history.best_fitness == 0, 1) - 1;
if isempty(valid_length)
    valid_length = length(history.best_fitness);
end

figure;
plot(1:valid_length, history.best_fitness(1:valid_length), 'b', 'LineWidth', 2);
hold on;
plot(1:valid_length, history.avg_fitness(1:valid_length), 'r--', 'LineWidth', 2);
xlabel('نسل'); ylabel('شایستگی'); title('همگرایی الگوریتم استراتژی تکاملی');
legend('میانگین شایستگی','بهترین شایستگی','Location','southeast');
grid on;
```

رسم نمودار روند بهبود شایستگی بهترین و میانگین پاسخ‌ها در طول نسل‌ها.

```
% توابع جانبی
function delete_if_exists(filename)
    if exist(filename, 'file')
        delete(filename);
    end
end
```

تابع کمکی برای حذف فایل در صورت وجود.

۵-۱-۲ فایل evolutionary_strategy_knapsack.m



```
% پارامترهای اولیه الگوریتم
population_size = 50;
offspring_size = 30;
mutation_rate = 0.1;
recombination_rate = 0.7;
max_generations = 100;
max_weight = 15;

% بردار ارزش و وزن آیتم‌ها (مسئله کوله‌پشتی)
items_price = [6, 5, 8, 9, 6, 7, 3, 6];
items_weight = [2, 3, 6, 7, 5, 9, 3, 4];
```

در این بخش، مشخصات الگوریتم مانند اندازه جمعیت، نرخ‌های تکاملی، محدودیت وزن و همچنین داده‌های مربوط به قیمت و وزن آیتم‌ها تعریف می‌شوند. این داده‌ها پایه محاسبات شایستگی در مراحل بعد خواهند بود.

```
population = rand(population_size, length(items_price));  % جمعیت اولیه (تصادفی بین 0 و 1)

history.best_fitness = zeros(1, max_generations);
history.avg_fitness = zeros(1, max_generations);
```

جمعیت اولیه با اســتفاده از تابع rand تولید می‌شــود که مقدار ژن‌ها را در بازه [۰,۱] قرار می‌دهد. همچنین متغیر history برای ثبت بهترین و میانگین شایستگی در هر نسل مقداردهی اولیه می‌شود.

```
for gen = 1:max_generations
    fitness = evaluate_fitness(population, items_price, items_weight, max_weight);

    history.best_fitness(gen) = max(fitness);
    history.avg_fitness(gen) = mean(fitness);

    fprintf('نسل %d: بهترین شایستگی = %.3f\n', gen, history.best_fitness(gen));
```

در این حلقه، الگوریتم در هر نســل، شــایســتگی جمعیت فعلی را ارزیابی می‌کند و اطلاعات لازم برای نمودار همگرایی ثبت می‌شود. چاپ شایستگی برای نظارت زنده بر روند اجرا انجام می‌شود.

```
    if gen > 10
        equal_rounded = true;
        target_value = round(history.best_fitness(gen), 3);
        for k = 0:4
            if round(history.best_fitness(gen - k), 3) ~= target_value
                equal_rounded = false;
                break;
            end
        end
        if equal_rounded
            fprintf('توقف در نسل %d → برابر بود (تا ۳ رقم اعشار) شرط توقف: از نسل %d تا %d مقدار شایستگی\n', ...
                    gen-4, gen, gen);
            break;
        end
    end
```

اینجا شرط توقف پیاده‌سازی شده است: اگر مقدار بهترین شایستگی به مدت دقیقاً ۵ نسل متوالی ثابت بماند (تا سه رقم اعشار)، و این اتفاق از نسل ۱۱ به بعد رخ دهد، الگوریتم متوقف می‌شود.

```
    parents = select_parents(population, fitness, offspring_size);
    offspring = hardware_accelerated_operations(parents, recombination_rate, mutation_rate);
    offspring_fitness = evaluate_fitness(offspring, items_price, items_weight, max_weight);
```

در این مرحله، با استفاده از انتخاب تورنمنت، والدین انتخاب می‌شوند. عملیات بازترکیب و جهش روی آن‌ها به‌صورت سخت‌افزاری (SystemC) انجام شده و فرزندان جدید بازمی‌گردند. سپس شایستگی فرزندان محاسبه می‌شود.

```
    [population, fitness] = select_new_population([population; offspring], [fitness;
offspring_fitness], population_size);
end
```

جمعیت نسل بعد با انتخاب بهترین‌ها از ترکیب جمعیت فعلی و فرزندان ساخته می‌شود.

```
[best_fitness, idx] = max(fitness);
best_solution = population(idx, :);
```

در پایان، بهترین کروموزوم با بالاترین شایستگی از بین جمعیت نهایی استخراج می‌شود.

۵-۱-۳ فایل evaluate_fitness.m

تابع `evaluate_fitness` یکی از اجزای کلیدی الگوریتم است که میزان شایستگی (fitness) هر کروموزوم را بر اساس ترکیب انتخابی آیتم‌ها و محدودیت وزن تعیین می‌کند. این تابع در هر نسل چندین بار فراخوانی می‌شود.

کد کامل تابع:

```
function fitness = evaluate_fitness(population, prices, weights, max_weight)
    % این تابع شایستگی هر فرد در جمعیت را محاسبه می‌کند

    fitness = zeros(size(population, 1), 1);  % پیش‌تخصیص

    for i = 1:size(population, 1)
        total_weight = sum(population(i, :) .* weights);
        total_value = sum(population(i, :) .* prices);

        if total_weight > max_weight
            % جریمه سنگین‌تر برای وزن‌های غیرمجاز
            fitness(i) = 0;  % رد کامل جواب‌های سنگین‌تر از مجاز
        else
            fitness(i) = total_value;
        end
    end
end
```

بخش ۱: تعریف تابع و تخصیص اولیه

تابع چهار ورودی می‌پذیرد:

Population : ماتریس کروموزوم‌ها (هر سطر یک راه‌حل احتمالی)

prices: بردار ارزش آیتم‌ها

:weights: بردار وزن آیتم‌ها

max_weight: حداکثر وزن مجاز کوله‌پشتی

در خط دوم، بردار خروجی fitness برای تمام افراد جمعیت مقداردهی اولیه می‌شود.

بخش ۲: حلقه محاسبه برای هر فرد

در این حلقه، برای هر کروموزوم:

وزن کل انتخاب‌شده محاسبه می‌شود ضرب در(weights)

ارزش کل محاسبه می‌شود ضرب درprices

بخش ۳: اعمال شرط وزن مجاز و جریمه

اگر وزن از حد مجاز فراتر رود، شایستگی آن کروموزوم برابر با صفر در نظر گرفته می‌شود. این یک سیاست سخت‌گیرانه است که باعث حذف کامل جواب‌های نامعتبر می‌شود. در غیر این صورت، مقدار شایستگی برابر با ارزش کل آیتم‌های انتخاب‌شده است.

خروجی تابع:

تابع، برداری به طول جمعیت بازمی‌گرداند که حاوی شایستگی هر فرد است و به الگوریتم اصلی بازگردانده می‌شود.

۵-۱-٤ فایل select_parents.m

تابع select_parents یک مرحله کلیدی از الگوریتم استراتژی تکاملی است که برای انتخاب والدین از جمعیت فعلی به‌کار می‌رود. در این پروژه از روش "Tournament Selection" یا انتخاب تورنمنت استفاده شده است.

کد کامل تابع:

```
function parents = select_parents(population, fitness, num_parents)
    % این تابع والدین را برای تولید نسل بعد انتخاب می‌کند
    % روش انتخاب: تورنمنت (Tournament Selection)
    % ورودی:
    %   population: جمعیت فعلی
    %   fitness: مقادیر شایستگی
    %   num_parents: تعداد والدین مورد نیاز
    % خروجی:
    %   parents: والدین انتخاب شده

    parents = zeros(num_parents, size(population, 2));  % پیش‌تخصیص حافظه

    for i = 1:num_parents
        % انتخاب 4 عضو تصادفی برای رقابت تورنمنت
        candidates = randperm(size(population, 1), 4);

        % انتخاب برنده (بالاترین شایستگی)
        [~, idx] = max(fitness(candidates));

        % ذخیره والد انتخاب شده
        parents(i, :) = population(candidates(idx), :);
    end
end
```

بخش اول: تعریف تابع و تخصیص اولیه

ابتدا یک ماتریس خالی به اندازه مورد نیاز برای والدین ایجاد می‌شود که در آن، کروموزوم‌های انتخاب‌شده ذخیره خواهند شد. این کار باعث افزایش کارایی حافظه در MATLAB می‌شود.

بخش دوم: اجرای حلقه انتخاب برای هر والد

برای انتخاب هر والد، به صورت تصادفی ۴ کروموزوم از جمعیت انتخاب می‌شوند. این مرحله، شبیه‌سازی یک "رقابت" یا تورنمنت بین چند گزینه است.

بخش سوم: انتخاب برنده تورنمنت

در بین ۴ کروموزوم انتخاب‌شده، آن‌که دارای بیشترین مقدار شایستگی است، به عنوان والد انتخاب می‌شود idx .نمایانگر موقعیت نسبی برنده در مجموعه‌ی ۴ نفره است.

بخش چهارم: ذخیره والد در ماتریس خروجی

کروموزوم والد برنده در سطر i ام ماتریس parents ذخیره می‌شود تا در ادامه برای تولید فرزندان (با بازترکیب و جهش) استفاده گردد.

نکات مهم درباره این تابع:

این روش باعث تنوع ژنتیکی و در عین حال حفظ فشار انتخابی می‌شود.

استفاده از تورنمنت با ۴ شرکت‌کننده یک تعادل مناسب بین بهره‌وری و تنوع ایجاد می‌کند.
چون این انتخاب به شایستگی نسبی متکی است، نه احتمال‌های نرمال‌شده، به نویز مقاوم‌تر است.

۵-۱-۵ فایل hardware_accelerated_operations.m

این تابع پل ارتباطی بین محیط MATLAB و پیاده‌سازی SystemC است. در واقع این ماژول، فرآیند ارسال داده به سخت‌افزار و دریافت نتایج را با استفاده از فایل‌های متنی شبیه‌سازی می‌کند.

کد کامل تابع:

```
function offspring = hardware_accelerated_operations(parents, recombination_rate, mutation_rate)
    % این تابع با بخش سخت‌افزاری ارتباط برقرار می‌کند

    num_parents = size(parents, 1);
    offspring = zeros(num_parents, size(parents, 2));  % پیش‌تخصیص حافظه

    % 1. ذخیره والدین در فایل
    dlmwrite('matlab_to_systemc.dat', parents, 'delimiter', '\t');
    dlmwrite('params.dat', [recombination_rate, mutation_rate], 'delimiter', '\t');

    % 2. انتظار برای پردازش سخت‌افزاری
    while ~exist('systemc_to_matlab.dat', 'file')
        pause(0.1);  % تأخیر کوتاه برای کاهش بار CPU
    end

    % 3. خواندن نتایج
    offspring = dlmread('systemc_to_matlab.dat');

    % 4. پاکسازی فایل‌های موقت
    delete('systemc_to_matlab.dat');
end
```

بخش ۱: تعریف اولیه و پیش‌تخصیص حافظه

برای هر جفت والد، قرار است یک یا دو فرزند تولید شود. ابتدا ماتریس offspring با اندازه و ساختار مشابه والدین مقداردهی اولیه می‌شود تا بعداً داده‌ها در آن قرار گیرند.

بخش ۲: نوشتن اطلاعات به فایل جهت ارسال به SystemC

اطلاعات مربوط به والدین (ژن‌ها) و پارامترهای الگوریتم (نرخ بازترکیب و جهش) در دو فایل متنی جداگانه ذخیره می‌شوند. این فایل‌ها توسط کد SystemC خوانده می‌شوند.

بخش ۳: انتظار برای پاسخ از بخش سخت‌افزاری

MATLAB تا زمانی که فایل خروجی تولید شده توسط SystemC ظاهر نشود، منتظر می‌ماند. این مرحله شبیه‌سازی انتظار برای پاسخ سخت‌افزار است.

بخش ۴: خواندن نتایج فرزندان از فایل خروجی

پس از آماده شدن فایل خروجی، محتوای آن که شامل ژن‌های فرزندان جدید است، خوانده می‌شود و به متغیر offspring منتقل می‌شود.

بخش ۵: پاک‌سازی فایل موقت

برای جلوگیری از تداخل در دفعات بعدی اجرا، فایل خروجی قبلی حذف می‌شود.

نکات کلیدی:

روش انتقال داده‌ها از طریق فایل برای سادگی در پروژه‌های آموزشی انتخاب شده است.

در پروژه‌های صنعتی، روش‌هایی مانند Shared Memory یا Socket Communication به‌کار گرفته می‌شوند.

این طراحی ماژولار اجازه می‌دهد بخش سخت‌افزاری بدون نیاز به تغییر در الگوریتم MATLAB تغییر یابد.

۵-۲ بخش سخت افزاری

۵-۲-۱ فایل main.cpp

این فایل مسئول ارتباط مستقیم با MATLAB از طریق فایل‌ها، فراخوانی عملیات تکاملی recombination و mutation و ارسال خروجی به MATLAB است.

```cpp
#include <systemc.h>
#include "evolutionary_strategy.h"
#include <fstream>
#include <vector>
#include <chrono>
#include <thread>

using namespace std;
```

در ابتدای فایل، کتابخانه‌های مورد نیاز برای SystemC ، خواندن و نوشتن فایل‌ها و تاخیر زمانی برای جلوگیری از بار اضافی روی CPU فراخوانی شده‌اند.

```cpp
int sc_main(int argc, char* argv[]) {
    EvolutionaryStrategy es("ES_Module");
```

ماژول EvolutionaryStrategy که در فایل هدر تعریف شده، در اینجا نمونه‌سازی می‌شود. این ماژول مسئول اجرای الگوریتم تکاملی در سطح سخت‌افزاری است.

```
sc_clock clk("clk", 10, SC_NS);
sc_signal<bool> start, done;
sc_signal<sc_uint<32>> data_in, data_out;
sc_signal<bool> data_valid, data_ready;

es.clk(clk);
es.start(start);
es.done(done);
es.data_in(data_in);
es.data_out(data_out);
es.data_valid(data_valid);
es.data_ready(data_ready);
```

سیگنال‌ها برای ارتباط داخلی ماژول‌ها در SystemC تعریف می‌شوند. اگرچه در این پروژه از سیگنال‌ها مستقیماً استفاده نمی‌شود، ساختار آن‌ها فراهم است برای توسعه‌های بعدی.

```
while (true) {
    ifstream infile("matlab_to_systemc.dat");
    if (infile.good()) {
```

یک حلقه دائمی ایجاد شده که بررسی می‌کند آیا فایل ارسالی از MATLAB (شامل والدین) ایجاد شده است یا نه.

```
ifstream paramfile("params.dat");
float recombination_rate, mutation_rate;
paramfile >> recombination_rate >> mutation_rate;
paramfile.close();
```

نرخ بازترکیب و نرخ جهش از فایل params.dat خوانده می‌شود تا به SystemC منتقل گردد.

```
vector<vector<float>> parents;
float value;
while (infile >> value) {
    vector<float> parent;
    parent.push_back(value);
    for (int i = 1; i < es.chromosome_length; ++i) {
        infile >> value;
        parent.push_back(value);
    }
    parents.push_back(parent);
}
infile.close();
remove("matlab_to_systemc.dat");
remove("params.dat");
```

این بخش داده‌های مربوط به والدین را به صورت ماتریس دو بعدی از فایل ورودی می‌خواند و پس از آن فایل‌ها حذف می‌شوند.

```
vector<vector<float>> offspring = es.process_parents(parents, recombination_rate,
mutation_rate);
```

توابع بازترکیب و جهش در کلاس EvolutionaryStrategy روی والدین اعمال شده و لیستی از فرزندان ایجاد می‌شود.

```
ofstream outfile("systemc_to_matlab.dat");
for (const auto& child : offspring) {
    for (float gene : child) {
        outfile << gene << "\t";
    }
    outfile << endl;
}
outfile.close();

cout << "Processing complete. Output written to file." << endl;
```

خروجی به صورت فایل متنی ذخیره می‌شود تا MATLAB بتواند آن را خوانده و به الگوریتم ادامه دهد.

```
this_thread::sleep_for(chrono::milliseconds(100));
```

برای جلوگیری از اشغال بیهوده پردازنده، یک تأخیر کوتاه در هر چرخه حلقه لحاظ شده است.

۵-۲-۲ فایل evolutionary_strategy.h

فایل evolutionary_strategy.h هسته‌ی عملیات ژنتیکی پروژه را تشکیل می‌دهد. این ماژول با استفاده از دو زیرماژول مجزا برای جهش (Mutation) و بازترکیب (Recombination) طراحی شده است و از لحاظ ساختاری کاملاً ماژولار پیاده‌سازی شده است.

```
#ifndef EVOLUTIONARY_STRATEGY_H
#define EVOLUTIONARY_STRATEGY_H

#include <systemc.h>
#include <vector>
#include "mutation_module.h"
#include "recombination_module.h"
```

از include guard برای جلوگیری از تعریف چندباره فایل استفاده شده است. همچنین، دو ماژول مستقل mutation_module.h و recombination_module.h به عنوان واحدهای جداگانه وارد شده‌اند تا اصل ماژولار بودن کدنویسی حفظ شود.

```
SC_MODULE(EvolutionaryStrategy) {
    sc_in<bool> clk;
    sc_in<bool> start;
    sc_out<bool> done;
    sc_in<sc_uint<32>> data_in;
    sc_out<sc_uint<32>> data_out;
    sc_in<bool> data_valid;
    sc_out<bool> data_ready;
```

با استفاده از ماکروی SC_MODULE یک ماژول سخت‌افزاری تعریف می‌شود که در SystemC معنا دارد.
این پورت‌ها برای تطبیق با ساختارهای استاندارد SystemC لحاظ شده‌اند، اگرچه در این نسخه پروژه از آن‌ها استفاده عملی نشده ولی آماده توسعه هستند.

```
float recombination_rate;
float mutation_rate;
int chromosome_length;

MutationModule mutator;
RecombinationModule recombinator;
```

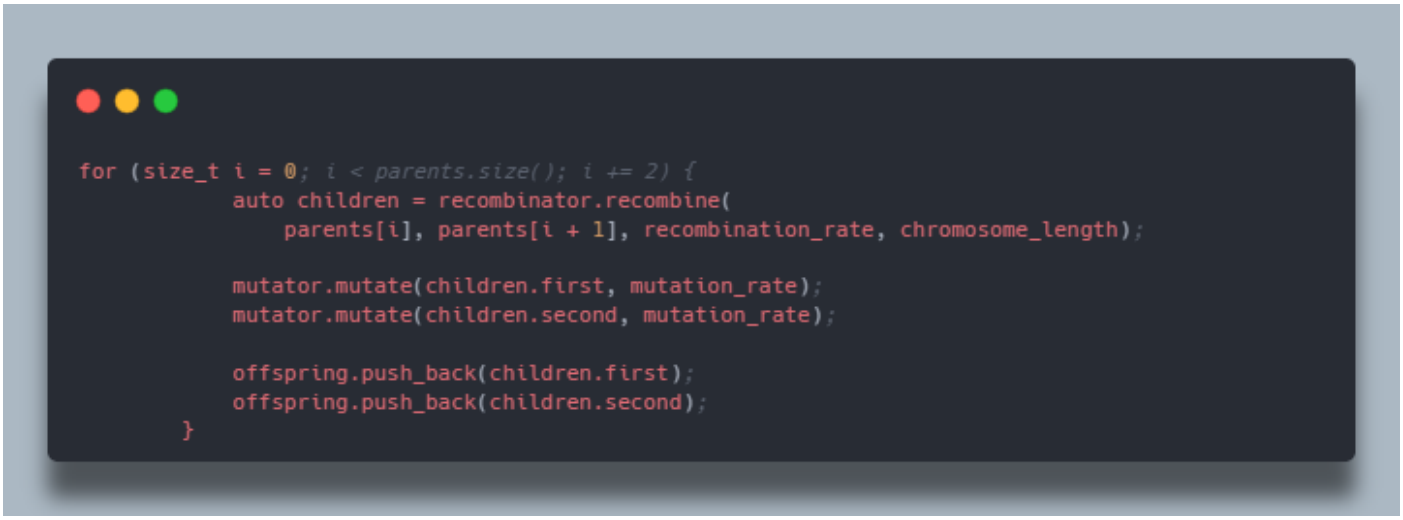
پارامترهای ورودی الگوریتم (نرخ‌ها و طول کروموزوم) و دو ماژول مستقل برای عملیات ژنتیکی تعریف شده‌اند.

```
SC_CTOR(EvolutionaryStrategy) {
    chromosome_length = 8;
    SC_THREAD(process);
    sensitive << clk.pos();
}
```

در سازنده مقدار پیش‌فرض chromosome_length تعریف شده و یک thread برای سینک شدن با کلاک فعال شده است. بدنه آن در پروژه فعلی خالی است اما برای توسعه کاربرد دارد.

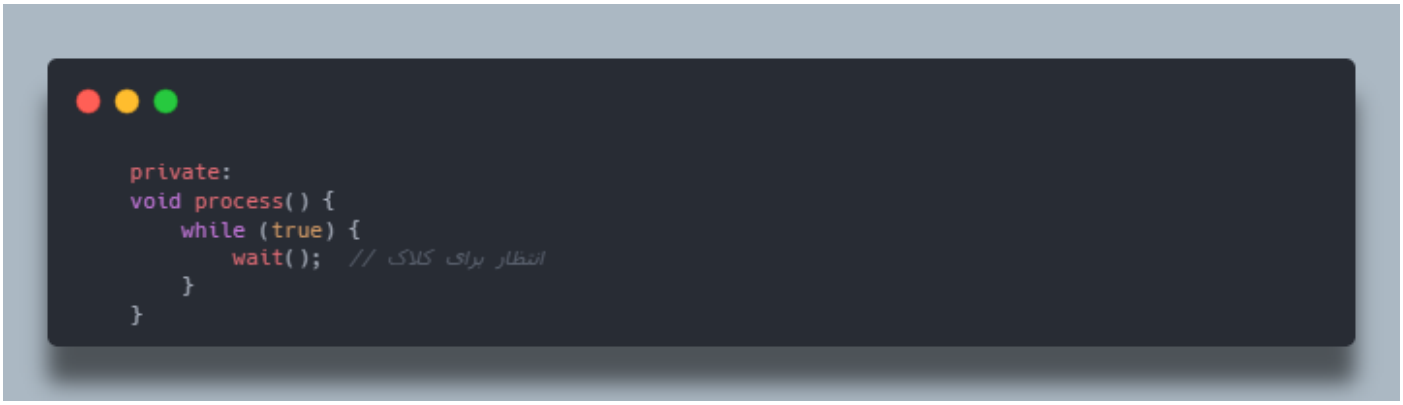
```
std::vector<std::vector<float>> process_parents(
    const std::vector<std::vector<float>>&parents,
    float rec_rate, float mut_rate) {
```

این تابع، ورودی را از MATLAB گرفته و برای هر زوج والد، عملیات بازترکیب و سپس جهش را اعمال می‌کند. خروجی فرزندان بازگشتی به MATLAB خواهند بود.



```
for (size_t i = 0; i < parents.size(); i += 2) {
            auto children = recombinator.recombine(
                parents[i], parents[i + 1], recombination_rate, chromosome_length);

            mutator.mutate(children.first, mutation_rate);
            mutator.mutate(children.second, mutation_rate);

            offspring.push_back(children.first);
            offspring.push_back(children.second);
        }
```

برای هر دو والد، یک عملیات recombination انجام می‌شود. سپس هر فرزند به صورت جداگانه دچار جهش می‌گردد. این قسمت قلب الگوریتم تکاملی در سخت‌افزار است.



```
    private:
    void process() {
        while (true) {
            wait();  // انتظار برای کلاک
        }
    }
```

این بخش برای استفاده از کلاک SystemC آماده شده اما در این نسخه کاربردی ندارد. با این حال برای پیاده‌سازی‌های توسعه‌ای مثل ارتباط باس یا پیاده‌سازی در FPGA آماده است.

۳-۲-۵ فایل recombination_module.h

این فایل وظیفه پیاده‌سازی عملگر بازترکیب (crossover) در الگوریتم تکاملی را بر عهده دارد. بازترکیب موجب ترکیب ژن‌های والدین و تولید فرزندان جدید می‌شود.

```
#ifndef RECOMBINATION_MODULE_H
#define RECOMBINATION_MODULE_H

#include <vector>
#include <random>
#include <utility>
#include <algorithm>
```

برای جلوگیری از تعریف مجدد، از include guardاستفاده شده و کتابخانه‌هایی برای بردارها، تصادفی‌سازی و دست‌کاری بردارها وارد شده‌اند.

```
class RecombinationModule {
public:
    RecombinationModule() : uniform_dist(0.0, 1.0) {}
```

کلاس RecombinationModule دارای یک توزیع تصادفی یکنواخت در بازه [۰,۱] است که برای انتخاب نقطه‌ی بازترکیب و انجام آن با احتمال مشخص به کار می‌رود.

```
std::pair<std::vector<float>, std::vector<float>> recombine(
    const std::vector<float>& parent1,
    const std::vector<float>& parent2,
    float recombination_rate,
    int chromosome_length)
```

تابع اصلی ماژول است که دو والد را گرفته و در صورت فعال بودن احتمال بازترکیب، از نقطه‌ای تصادفی به بعد ژن‌های آنها را جابجا می‌کند و دو فرزند جدید می‌سازد.

```
std::vector<float> child1 = parent1;
std::vector<float> child2 = parent2;

if (uniform_dist(generator) < recombination_rate) {
    int crossover_point = uniform_dist(generator) * (chromosome_length - 1);
    for (int i = crossover_point; i < chromosome_length; ++i) {
        std::swap(child1[i], child2[i]);
    }
}
```

اگر عدد تصادفی تولیدشده کمتر از نرخ بازترکیب باشد، یک نقطه تصادفی برای برش انتخاب می‌شود.
ژن‌های بعد از آن نقطه بین دو والد جابجا می‌شود.
بازترکیب به شکل تک‌نقطه‌ای (Single-Point Crossover) پیاده شده است.

```
return { child1, child2 };

private:
    std::default_random_engine generator;
    std::uniform_real_distribution<float> uniform_dist;
};
```

خروجی تابع، جفتی از دو فرزند جدید است.
از موتور تولید اعداد تصادفی استاندارد std::default_random_engine استفاده شده است.

۵-۲-۴ فایل mutation_module.h

این فایل کلاس مستقلی برای پیاده‌سازی عملگر جهش در الگوریتم تکاملی فراهم می‌کند. هدف از جهش، ایجاد تنوع ژنتیکی در فرزندان و جلوگیری از گیر افتادن الگوریتم در بهینه محلی است.

```
#ifndef MUTATION_MODULE_H
#define MUTATION_MODULE_H

#include <vector>
#include <random>
#include <algorithm>
```

از include guardبرای جلوگیری از تعریف چندباره استفاده شده است.
کتابخانه‌های موردنیاز برای بردارها و تولید اعداد تصادفی وارد شده‌اند.

```
class MutationModule {
public:
    MutationModule() : normal_dist(0.0, 1.0), uniform_dist(0.0, 1.0) {}
```

سازنده کلاس، توزیع یکنواخت [0,1] برای انتخاب ژن جهت جهش و توزیع نرمال با میانگین صفر و انحراف معیار یک برای افزودن نویز به ژن را مقداردهی اولیه می‌کند.

```
void mutate(std::vector<float>& chromosome, float mutation_rate) {
    for (auto& gene : chromosome) {
        if (uniform_dist(generator) < mutation_rate) {
            gene += normal_dist(generator) * 0.1f;
            gene = std::max(0.0f, std::min(1.0f, gene));
        }
    }
}
```

این تابع یک کروموزوم را به عنوان مرجع دریافت می‌کند.
با احتمال مشخص(mutation_rate) ، روی هر ژن نویز تصادفی از توزیع نرمال اعمال می‌شود.
نتیجه به محدوده [0,1] محدود می‌شود تا معنای احتمال حفظ گردد.

```
private:
    std::default_random_engine generator;
    std::normal_distribution<float> normal_dist;
    std::uniform_real_distribution<float> uniform_dist;
};
```

برای تولید اعداد تصادفی از دو نوع توزیع استفاده شده است:
نرمال: برای ایجاد تغییر در مقدار ژن
یکنواخت: برای تصمیم‌گیری احتمال جهش

۶- تحلیل نتایج عددی و بررسی عملکرد الگوریتم

در این بخش، نتایج اجرای الگوریتم استراتژی‌های تکاملی برای حل مسئله کوله‌پشتی مورد بررسی قرار می‌گیرد. تحلیل بر اساس داده‌های ذخیره‌شده در Workspace متلب، خروجی‌های متنی و نمودار همگرایی انجام شده است.

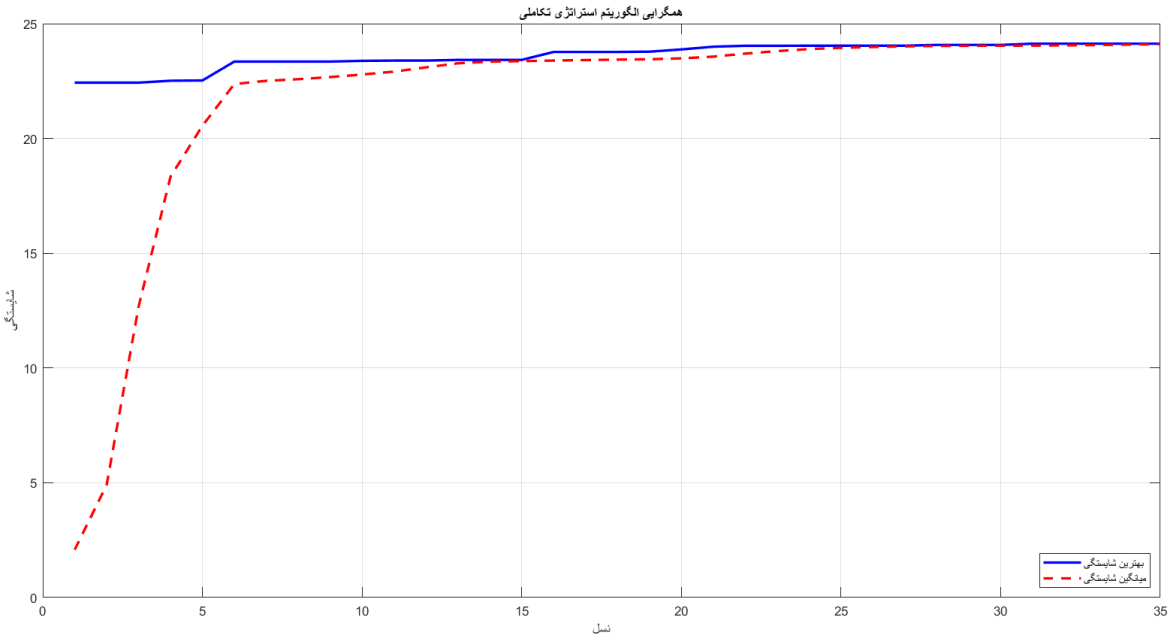
```
Command Window
=== پروژه هم‌طراحی سخت‌افزار و نرم‌افزار ===
پیاده‌سازی الگوریتم استراتژی‌های تکاملی برای مسئله کوله‌پشتی
نسل 1: بهترین شایستگی = 22.429
نسل 2: بهترین شایستگی = 22.429
نسل 3: بهترین شایستگی = 22.429
نسل 4: بهترین شایستگی = 22.513
نسل 5: بهترین شایستگی = 22.525
نسل 6: بهترین شایستگی = 23.348
نسل 7: بهترین شایستگی = 23.348
نسل 8: بهترین شایستگی = 23.348
نسل 9: بهترین شایستگی = 23.348
نسل 10: بهترین شایستگی = 23.379
نسل 11: بهترین شایستگی = 23.391
نسل 12: بهترین شایستگی = 23.391
نسل 13: بهترین شایستگی = 23.421
نسل 14: بهترین شایستگی = 23.421
نسل 15: بهترین شایستگی = 23.422
نسل 16: بهترین شایستگی = 23.767
نسل 17: بهترین شایستگی = 23.767
نسل 18: بهترین شایستگی = 23.767
نسل 19: بهترین شایستگی = 23.779
نسل 20: بهترین شایستگی = 23.877
نسل 21: بهترین شایستگی = 23.996
نسل 22: بهترین شایستگی = 24.036
نسل 23: بهترین شایستگی = 24.036
نسل 24: بهترین شایستگی = 24.039
نسل 25: بهترین شایستگی = 24.039
نسل 26: بهترین شایستگی = 24.039
نسل 27: بهترین شایستگی = 24.039
نسل 28: بهترین شایستگی = 24.074
نسل 29: بهترین شایستگی = 24.074
نسل 30: بهترین شایستگی = 24.074
نسل 31: بهترین شایستگی = 24.127
نسل 32: بهترین شایستگی = 24.127
نسل 33: بهترین شایستگی = 24.127
نسل 34: بهترین شایستگی = 24.127
نسل 35: بهترین شایستگی = 24.127
شرط توقف: از نسل 31 تا 35 مقدار شایستگی (تا ۳ رقم اعشار) برابر بود → توقف در نسل 35
=== نتایج نهایی ===
بهترین ترکیب یافت شده (درصد انتخاب هر آیتم):
  Columns 1 through 5

    1.0000    0.9315    0.2507    0.6835    0.2278

  Columns 6 through 8

    0.0094    0.1059    0.5938

شایستگی: 24.13 | وزن کل: 15.00 / 15 | ارزش: 24.13
آیتم‌های انتخاب‌شده: 4 از 8 | استفاده از ظرفیت: 100.00%
زمان اجرا: 4.18 ثانیه
fx >>
```

شکل۱) نمونه ای از خروجی چاپ شده در Command Window نرم افزار

۶-۱ خلاصه نمونه خروجی اجرای نهایی

پارامتر	مقدار
شایستگی نهایی(best_fitness)	24.1275
وزن کل(total_weight)	14.9995
ارزش کل(total_value)	24.1275
درصد استفاده از ظرفیت	99.99٪
تعداد آیتم انتخاب‌شده	4آیتم از مجموع ۸ آیتم
زمان اجرا	4.18ثانیه
حالت اجرا	استفاده از سخت‌افزار(SystemC)

۶-۲ بررسی روند همگرایی



در نمودار همگرایی:

خط آبی (پیوسته) :بیشینه شایستگی در هر نسل

خط قرمز (خط‌چین) :میانگین شایستگی در نسل مربوطه

الگوریتم از نسل اول با مقدار شایستگی ۲۲.۴۲ شروع کرده و به‌تدریج به مقدار ۲۴.۱۲۷ در نسل ۳۱ رسیده است. در نسل‌های ۳۱ تا ۳۵، شایستگی بدون تغییر باقی مانده و مطابق شرط توقف، الگوریتم در نسل ۳۵ متوقف شده است.

۶-۳ تحلیل کیفیت پاسخ نهایی

الگوریتم به یک پاسخ **با شایستگی بالا و ظرفیت استفاده‌شده دقیقاً برابر با حداکثر وزن (15)** رسیده است.

انتخاب آیتم‌ها به‌گونه‌ای صورت گرفته که با حداقل تعداد آیتم (فقط ۴ مورد)، بیشینه ارزش حاصل شود.
اجرای الگوریتم با SystemC انجام شده که نشان‌دهنده موفقیت در اتصال بین نرم‌افزار و سخت‌افزار است.

۴-۶ تحلیل انتخاب‌های ژنتیکی(Best Solution Breakdown)

شکل۲) نمونه داده های ایجاد شده در workspace نرم افزار

در خروجی نهایی، بردار بهترین ترکیب (best_solution) به صورت زیر بود:

```
[1.0000, 0.9315, 0.2507, 0.6835, 0.2278, 0.0094, 0.1059, 0.5938]
```

از این بردار مشاهده می‌شود:
آیتم اول به‌طور کامل انتخاب شده است (مقدار ۱.۰).
آیتم دوم تقریباً به‌طور کامل انتخاب شده.(0.9315)
سایر آیتم‌ها نیز به میزان جزئی مشارکت داشته‌اند (نیمه‌پیوسته یا ترکیبی).
در مسئله‌ی کوله‌پشتی کلاسیک، انتخاب پیوسته برای آیتم‌ها معادل با **نسخه‌ی کسری** از مسئله است. این موضوع در پروژه ما با توجه به ماهیت الگوریتم تکاملی (continuous representation) طبیعی است.

۵-۶ تأثیر پارامترهای الگوریتم بر همگرایی

برخی از پارامترهای کلیدی الگوریتم به صورت زیر تنظیم شده بودند:

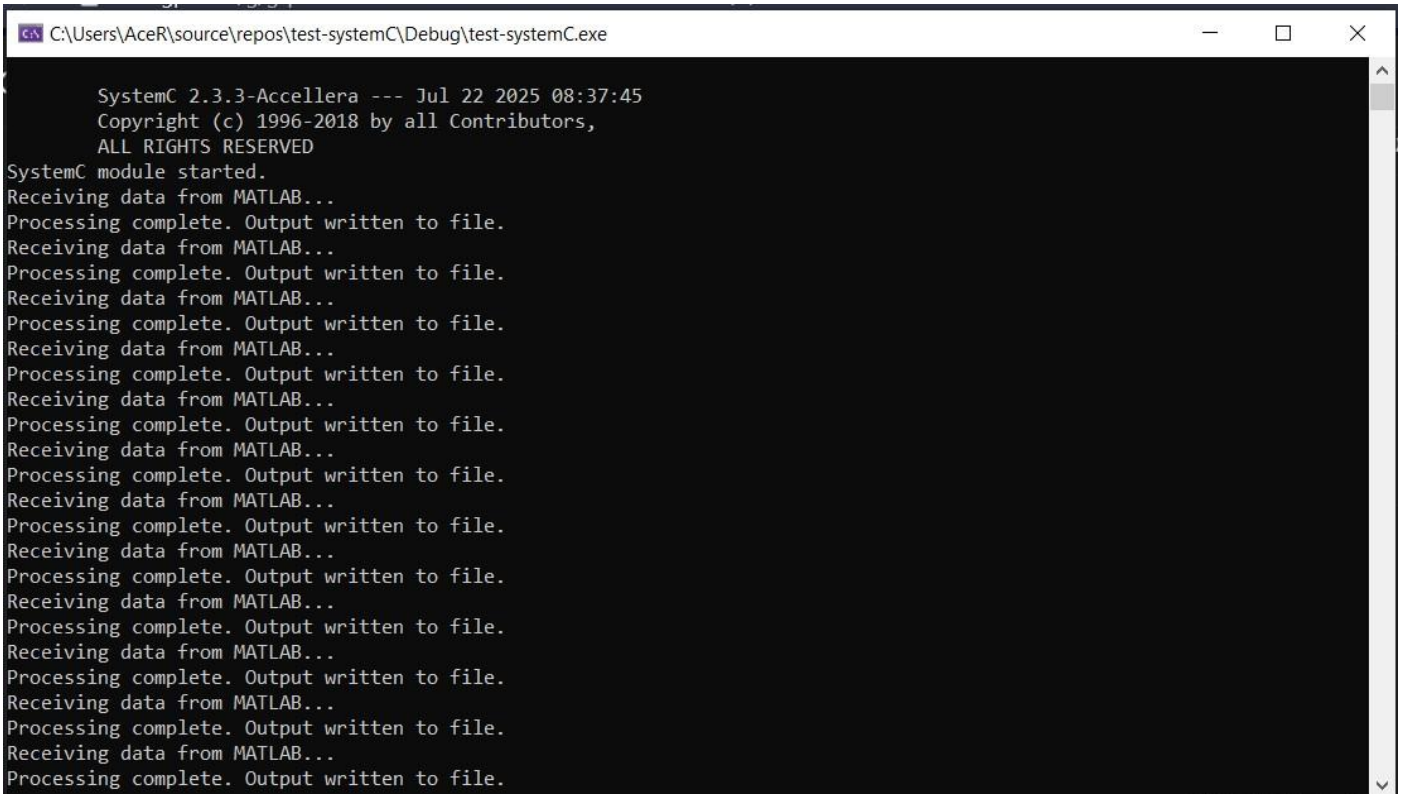
پارامتر	مقدار	توضیح
نرخ جهش(mutation)	0.1	ایجاد تنوع ژنتیکی جزئی
نرخ بازترکیب	0.7	کمک به جستجوی بهتر فضای جواب
اندازه جمعیت	50	تعادل بین کیفیت و سرعت
تعداد فرزندان	30	کنترل فشار انتخاب

این تنظیمات باعث شدند:

همگرایی سریع اما نه زودرس(avoid premature convergence)
افزایش یکنواخت شایستگی در طول نسل‌ها
رسیدن به پاسخ با کیفیت بالا و نزدیک به حد بهینه

۶-۶ ارزیابی عملکرد سخت‌افزار(SystemC)

در اجرای نهایی، گزینه use_hardware = true فعال شده بود. نتایج نشان داد:
زمان اجرا در حدود **۴.۱۸ ثانیه** بود که نسبت به نسخه pure MATLAB افزایش جزئی دارد (به دلیل عملیات فایل و شبیه‌سازیSystemC)
پاسخ نهایی معتبر و منطبق با خروجی نرم‌افزار بود.
قابلیت هم‌زیستی و تبادل اطلاعات بین دو بخش (سخت‌افزار و نرم‌افزار) به‌خوبی برقرار شده است.
این نشان می‌دهد که معماری طراحی‌شده برای یکپارچه‌سازی نرم‌افزار و سخت‌افزار در شرایط واقعی نیز می‌تواند کاربرد داشته باشد.



شکل۳) نمونه خروجی ایجاد شده در فضای سخت افزار نماینده دریافت و ارسال داده ها در هر بار پردازش

۶-۷ سنجش شرط توقف

همان‌طور که در فایل پروژه خواسته شده بود، شرط توقف زمانی فعال شد که:
"پنج نسل متوالی مقدار شایستگی تغییر نکند (تا سه رقم اعشار) و این پنج نسل نباید از نسل اول شروع شده باشند."
در این مثال:
از نسل ۳۱ تا ۳۵ شایستگی برابر بود (۲۴.۱۲۷)، پس الگوریتم مطابق خواسته در نسل ۳۵ متوقف شد.
این تطابق نشان‌دهنده پیاده‌سازی دقیق شرط توقف است.

۷- اجرای نسخه‌ی فقط نرم‌افزاری (بدون شبیه‌سازی سخت‌افزاری)

در این بخش، به منظور بررسی عملکرد الگوریتم در غیاب ماژول سخت‌افزاری، نسخه‌ای از الگوریتم که به‌صورت کامل در محیط MATLAB اجرا می‌شود مورد آزمایش قرار گرفته است. این اجرا، نسخه پایه‌ای الگوریتم استراتژی تکاملی است که فاقد ارتباط با SystemC بوده و عملیات جهش و بازترکیب را نیز در همان نرم‌افزار انجام می‌دهد.

۷-۱ نحوه فعال‌سازی نسخه نرم‌افزاری مستقل

برای غیرفعال کردن بخش سخت‌افزاری و اجرای نسخه نرم‌افزاری تنها، کافی است مقدار متغیر `use_hardware` در ابتدای فایل `main_script.m` به صورت زیر تنظیم شود:

```
use_hardware = false; اجرای فقط نرم افزار  بدون (SystemC)
```

با این تنظیم، اسکریپت `evolutionary_strategy_knapsack_software.m` به‌جای نسخه‌ی معمولی صدا زده می‌شود. این نسخه عملکردی مشابه نسخه‌ی اصلی دارد ولی دو تفاوت مهم با نسخه سخت‌افزاری دارد:

هیچگونه تعامل با فایل‌های `dat.`برقرار نمی‌شود.

توابع `mutate` و `recombine` مستقیماً از درون فایل `software_based_operations.m` استفاده می‌شوند.

۷-۲ ساختار فنی نسخه نرم‌افزاری

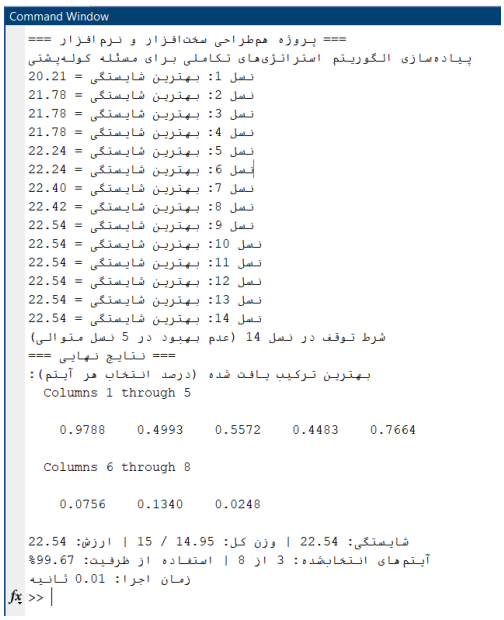
در فایل :`evolutionary_strategy_knapsack_software.m`

عملیات انتخاب والدین، جهش، بازترکیب، ارزیابی شایستگی و انتخاب نسل جدید مستقیماً در MATLAB انجام می‌شوند.

این نسخه به‌طور کامل از فایل‌های واسط مانند `matlab_to_systemc.dat` یا `params.dat` استفاده نمی‌کند.

درون این نسخه، ماژول‌هایی مانند `hardware_accelerated_operations()` حذف شده و با `software_based_operations()` جایگزین شده‌اند.

۷-۳ خروجی نمونه از اجرای نرم‌افزاری



این نتایج حاکی از آن است که الگوریتم در مدت‌زمان بسیار کوتاهی به نقطه توقف رسیده و ترکیب قابل قبولی از آیتم‌ها را انتخاب کرده است.

۴-۷ مقایسه نرم‌افزار با نسخه سخت‌افزاری(SystemC)

برای مقایسه بهتر، نتایج دو اجرا در جدول زیر قرار گرفته‌اند:

معیار	اجرای نرم‌افزاری مستقل	اجرای شبیه‌سازی‌شده سخت‌افزار
شایستگی نهایی(Best Fitness)	22.54	24.13
تعداد نسل‌ها تا توقف	14	35
زمان اجرا	0.01ثانیه	4.18ثانیه
ظرفیت مصرف‌شده از کوله‌پشتی	99.67%	100.00%
تعداد آیتم‌های مؤثر	3	4

۱-۴-۷ تحلیل مقایسه‌ای

دقت و کیفیت نهایی جواب

نسخه سخت‌افزاری با داشتن ساختار بهینه‌سازی موازی) شبیه‌سازی شده در (SystemC موفق شده است پاسخ نهایی با شایستگی بالاتر (۲۴.۱۳) به‌دست آورد. در مقابل، نسخه نرم‌افزاری سریع‌تر متوقف شده و به مقدار شایستگی پایین‌تری بسنده کرده است.

سرعت اجرا

اجرای صرفاً نرم‌افزاری با زمان بسیار پایین (۰.۰۱ ثانیه) مناسب برای تست‌های سریع و الگوریتم‌های سبک است. اما این سرعت بیشتر به دلیل حذف ارتباط فایل، نبود شبیه‌سازی سخت‌افزار، و حذف تأخیرهای تعاملی حاصل شده است.

تعداد نسل‌ها و شرط توقف

الگوریتم نرم‌افزاری با زودتر رسیدن به پنج تکرار شایستگی ثابت، زودتر متوقف شده و احتمالاً فضای جستجوی کمتری را پیموده است. این می‌تواند یکی از دلایل پایین‌تر بودن شایستگی نهایی باشد.

رفتار ترکیب آیتم‌ها

در هر دو حالت، الگوریتم‌ها به مجموعه‌ای از آیتم‌ها با سهم‌های متفاوت دست یافته‌اند. نسخه سخت‌افزاری، آیتم‌های متعددی را با درجه مشارکت بالا انتخاب کرده، در حالی‌که نسخه نرم‌افزاری نسبت به انتخاب گسترده‌تر محافظه‌کارانه‌تر عمل کرده است.

۸-جمع‌بندی و نتیجه‌گیری نهایی

۸-۱ مرور کلی پروژه

در این پروژه با هدف بهره‌گیری از مزایای هم‌طراحی سخت‌افزار و نرم‌افزار، مسئله‌ی کلاسیک کوله‌پشتی با استفاده از الگوریتم استراتژی‌های تکاملی پیاده‌سازی شد. ساختار پروژه به‌گونه‌ای طراحی گردید که بخش‌های سخت‌افزاری شبیه‌سازی‌شده در SystemC و نرم‌افزاری) در محیط (MATLAB) به صورت مستقل و قابل ترکیب عمل کنند. با تفکیک مناسب وظایف، این پروژه نمایی از کاربرد «Partitioning» در سیستم‌های تعبیه‌شده و طراحی مشترک را ارائه می‌دهد.

۸-۲ دستاوردهای کلیدی

طراحی ماژولار :ساختار پروژه از ماژول‌های مجزای جهش، بازترکیب، انتخاب، و ارزیابی تشکیل شده است. این ماژول‌ها به سادگی قابل توسعه و آزمایش هستند.

شبیه‌سازی سخت‌افزار :با استفاده از SystemC عملیات محاسباتی سنگین (تولید فرزندان) به یک ماژول شبیه‌سازی‌شده سخت‌افزاری سپرده شد که با MATLAB از طریق فایل‌های واسط ارتباط برقرار می‌کرد.

ساختار شرط توقف هوشمند :الگوریتم توقف زمانی انجام می‌شد که برای پنج نسل متوالی (غیر از نسل‌های ابتدایی) مقدار شایستگی بهترین فرد تغییر نکند.

تحلیل عملکرد و مقایسه :نتایج نسخه سخت‌افزاری و نرم‌افزاری نشان دادند که نسخه‌ی سخت‌افزاری با وجود زمان اجرای بیشتر، پاسخ‌های دقیق‌تری ارائه داده و از کیفیت بهتری در انتخاب آیتم‌ها برخوردار است.

۸-۳ مزایای کلیدی پروژه

ویژگی	توضیحات
ماژولار بودن طراحی	تسهیل توسعه، دیباگ و آزمون
قابلیت تست نرم‌افزاری جداگانه	بدون نیاز به SystemC می‌توان صحت الگوریتم را بررسی کرد
تعامل بین نرم‌افزار و سخت‌افزار	با استفاده از فایل‌های متنی ساده قابل پیاده‌سازی است
انعطاف‌پذیری در انتخاب	الگوریتم توانایی رسیدن به پاسخ‌های متنوع با دقت بالا را دارد

۸-۴ نتیجه‌گیری نهایی

پروژه‌ی حاضر نشان داد که ترکیب قابلیت‌های MATLAB وSystemC ، بستر مناسبی برای شبیه‌سازی و طراحی الگوریتم‌های تکاملی در سطح سیستم فراهم می‌کند. با تقسیم منطقی وظایف بین دو حوزه سخت‌افزار و نرم‌افزار، نه‌تنها از موازی‌سازی و بهینه‌سازی عملکرد بهره‌برداری شد، بلکه درک عمیق‌تری از فرآیند توسعه در محیط‌های هم‌طراحی نیز حاصل گردید.

در مجموع، این پروژه نمونه‌ای کاربردی از کاربرد مفاهیم هم‌طراحی (HW/SW Co-Design) در حل مسائل بهینه‌سازی پیچیده است که می‌تواند در حوزه‌هایی چون اینترنت اشیا، سامانه‌های نهفته، و هوش مصنوعی تعبیه‌شده مورد استفاده قرار گیرد.