به نام خدا

گزارش پروژه شماره ۱۹ درس سیستم‌های بی‌درنگ
تیم شماره ۱۵، محمد حسین دولت‌آبادی و محمد حیدری

توضیحات پروژه

ابتدا به توضیح الگوریتم‌های مطرح شده می‌پردازیم:

الگوریتم Chaos Game Optimization

1. تعریف مسئله بهینه‌سازی: ابتدا باید مسئله بهینه‌سازی خود را تعریف کنید. این شامل تعریف تابع هدف و محدودیت‌ها است.
2. طراحی فضای جستجو: بر اساس مسئله بهینه‌سازی، باید فضای جستجوی مناسب را طراحی کنید. این شامل تعیین متغیرها و محدوده‌های آنها است.
3. تعیین تابع ارزیابی: برای ارزیابی هر حالت در فضای جستجو، باید یک تابع ارزیابی (fitness function) تعیین کنید. این تابع بر اساس تابع هدف و محدودیت‌ها، عملکرد هر حالت را ارزیابی می‌کند.
4. پیاده‌سازی الگوریتم CGO: بر اساس الگوریتم CGO، باید یک روش برای جستجو در فضای جستجوی تعریف شده را پیاده‌سازی کنید. این شامل انتخاب نقاط شروع تصادفی، حرکت نقاط به سمت نقاط دیگر و بهبود تدریجی جواب است.
5. اجرای الگوریتم و بهبود جواب: با اجرای الگوریتم CGO، می‌توانید بهبود تدریجی جواب را مشاهده کنید. می‌توانید الگوریتم را تا زمانی که جواب بهینه‌ای به دست آید، اجرا کنید.

الگوریتم African Vultures Optimization

الگوریتم بهینه‌سازی گرگ‌های ولچر آفریقایی (African Vultures Optimization) یک الگوریتم بهینه‌سازی است که بر اساس رفتار گرگ‌های ولچر آفریقایی الهام گرفته شده است. این الگوریتم برای حل مسائل بهینه‌سازی مورد استفاده قرار می‌گیرد.

در این الگوریتم، رفتار گرگ‌های ولچر آفریقایی که به صورت گروهی شکار می‌کنند، مدلسازی می‌شود. این گروه از گرگ‌ها در جستجوی غذا به صورت هماهنگ عمل می‌کنند و با استفاده از تکنیک‌های خاصی، بهینه‌سازی را انجام می‌دهند.

الگوریتم African Vultures Optimization به صورت زیر عمل می‌کند:

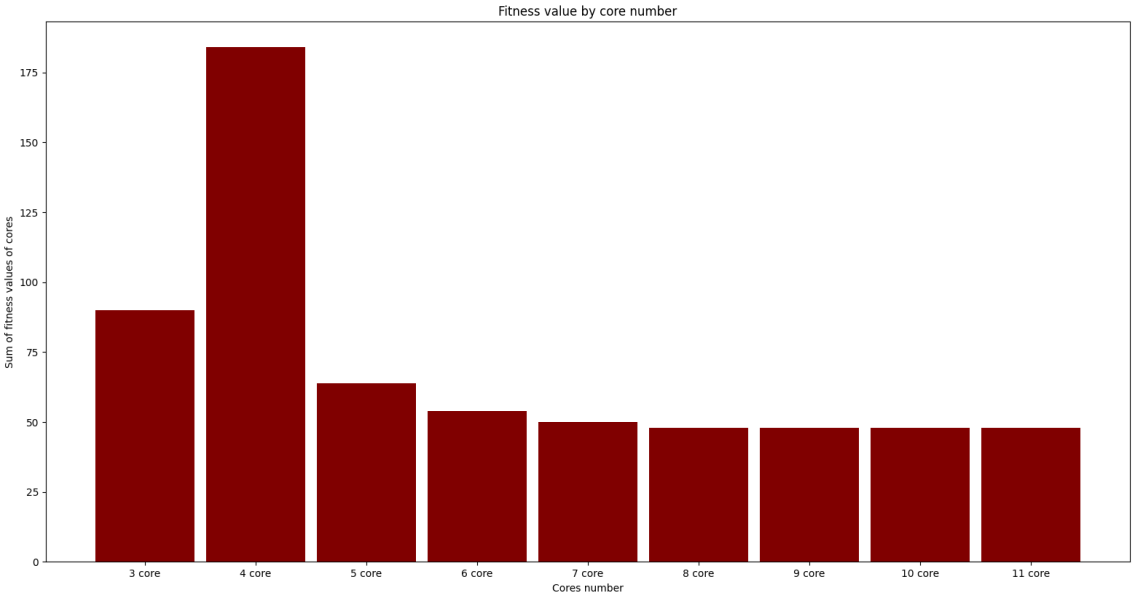
1. مرحله شکار: در این مرحله، گروهی از گرگ‌ها به صورت تصادفی در فضای جستجو حرکت می‌کنند. هر گرگ با استفاده از تکنیک‌های خاصی، موقعیت خود را بهبود می‌بخشد.
2. مرحله تبادل اطلاعات: در این مرحله، گرگ‌ها اطلاعات خود را با یکدیگر به اشتراک می‌گذارند. این اطلاعات شامل موقعیت‌ها و عملکرد گرگ‌ها است.
3. مرحله تصمیم‌گیری: در این مرحله، گرگ‌ها بر اساس اطلاعات دریافتی و تجربه خود، تصمیم‌گیری می‌کنند که در کدام نقطه از فضای جستجو حرکت کنند.
4. مرحله بهبود: در این مرحله، گرگ‌ها با استفاده از تکنیک‌های خاصی، موقعیت خود را بهبود می‌بخشند و بهینه‌سازی را انجام می‌دهند.
5. این مراحل به صورت تکراری ادامه می‌یابند تا به یک جواب بهینه نزدیک شود.

در این پروژه ما به پیاده‌سازی این دو الگوریتم که توضیح داده شد می‌پردازیم. برای بررسی حالت‌های مختلف، نیاز به توابع fitness که در داک به آن‌ها اشاره شده داریم. با توجه به داک، ما fitness های زیر را پیاده‌سازی کرده‌ایم. لازم به ذکر است که برخی از این توابع، در داک نیامده است و اضافه هستند.

```
class FitnessApproach(Enum):
    ONLY_DEADLINE = 1
    RESPONSE_TIME = 2
    WAIT_TIME = 3
    COMPLETION_TIME = 4
    LATENCY_TIME = 5
    SLACK_TIME = 6
    CUSTOM = 7
```

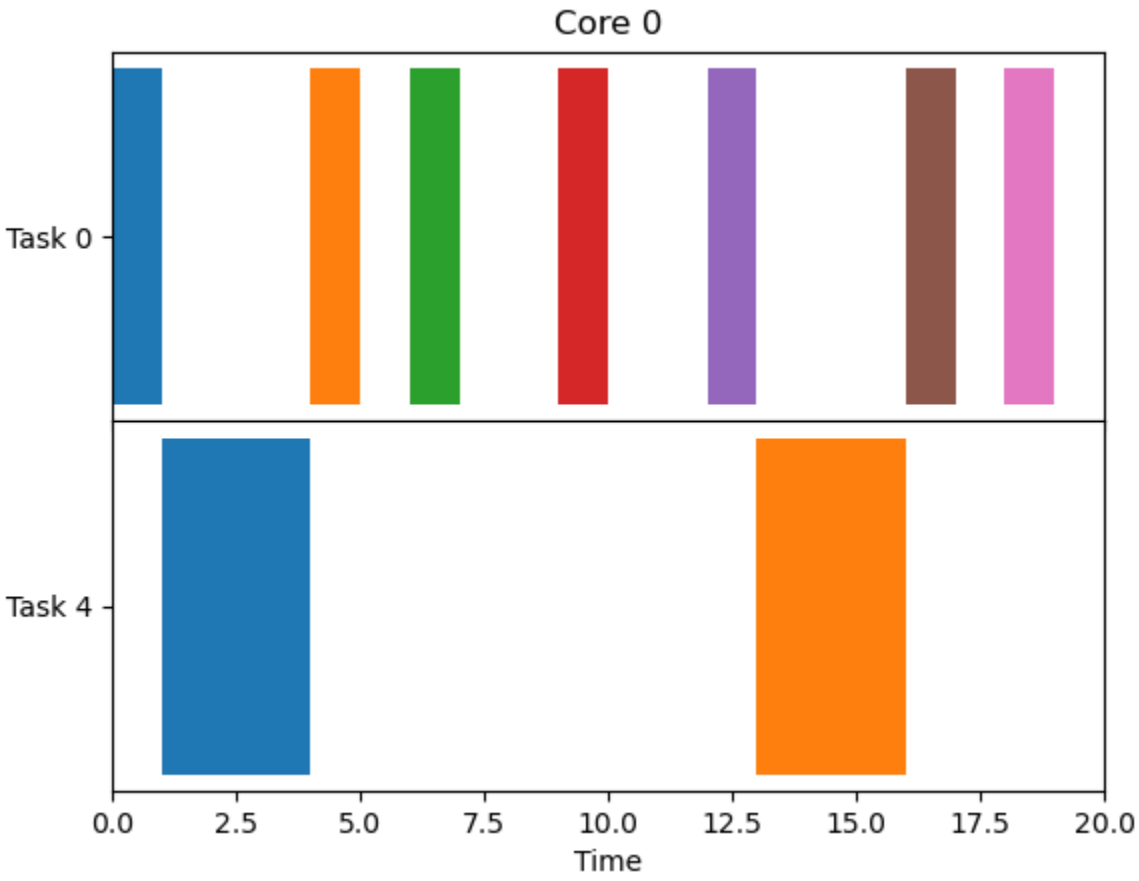
در تصویر بالا تمام توابع fitness که پیاده‌سازی شده‌اند آمده است.
نوع اول بررسی می‌کند که ددلاین‌ها حتما رعایت شود. این تابع علاوه بر آنکه به تنهایی می‌تواند به کار گرفته شود، با توجه به ماهیت بی‌درنگ پروژه در مابقی توابع نیز به کار برده شده اند چرا که همواره نیاز است تا ددلاین تسک‌ها رعایت شوند.

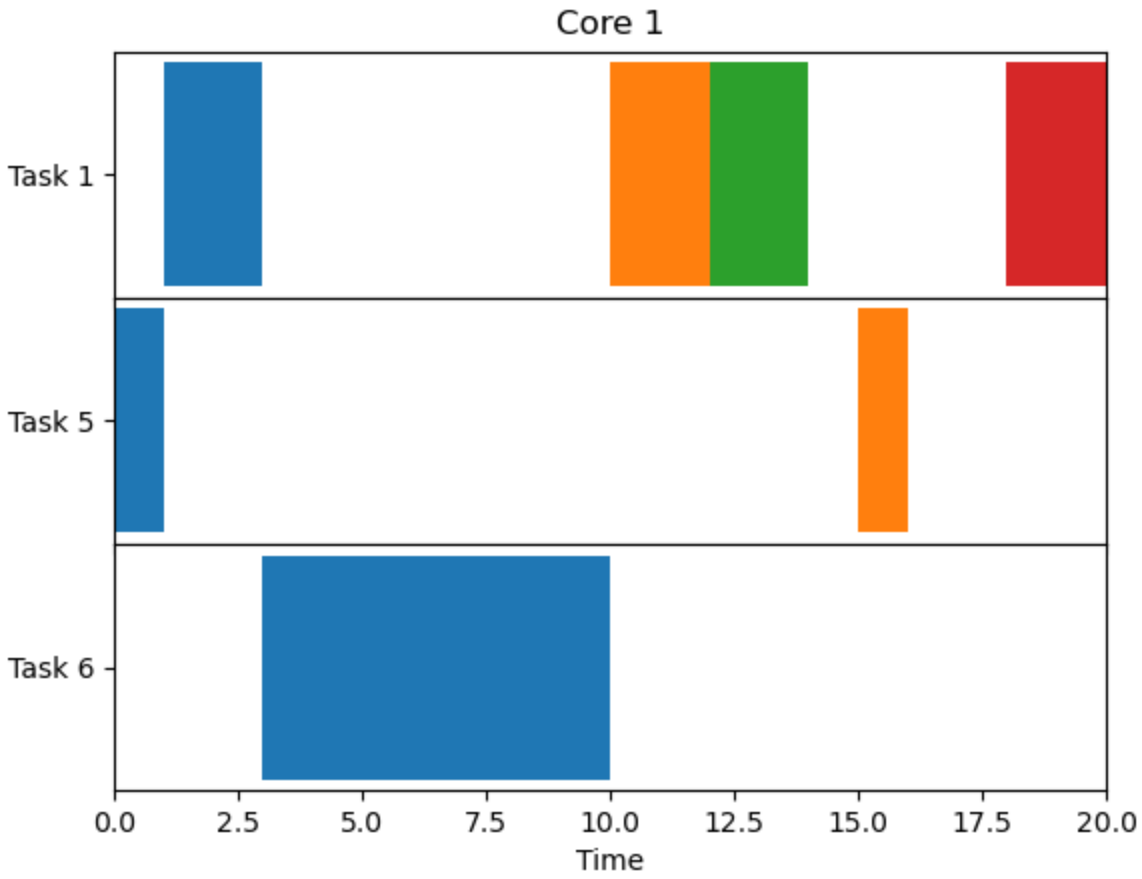
مورد دوم سعی در بهینه کردن زمان پاسخ‌دهی یا response time دارند. همانطور که اشاره شد در این تابع نیز شرط رعایت ددلاین‌ها برقرار است.

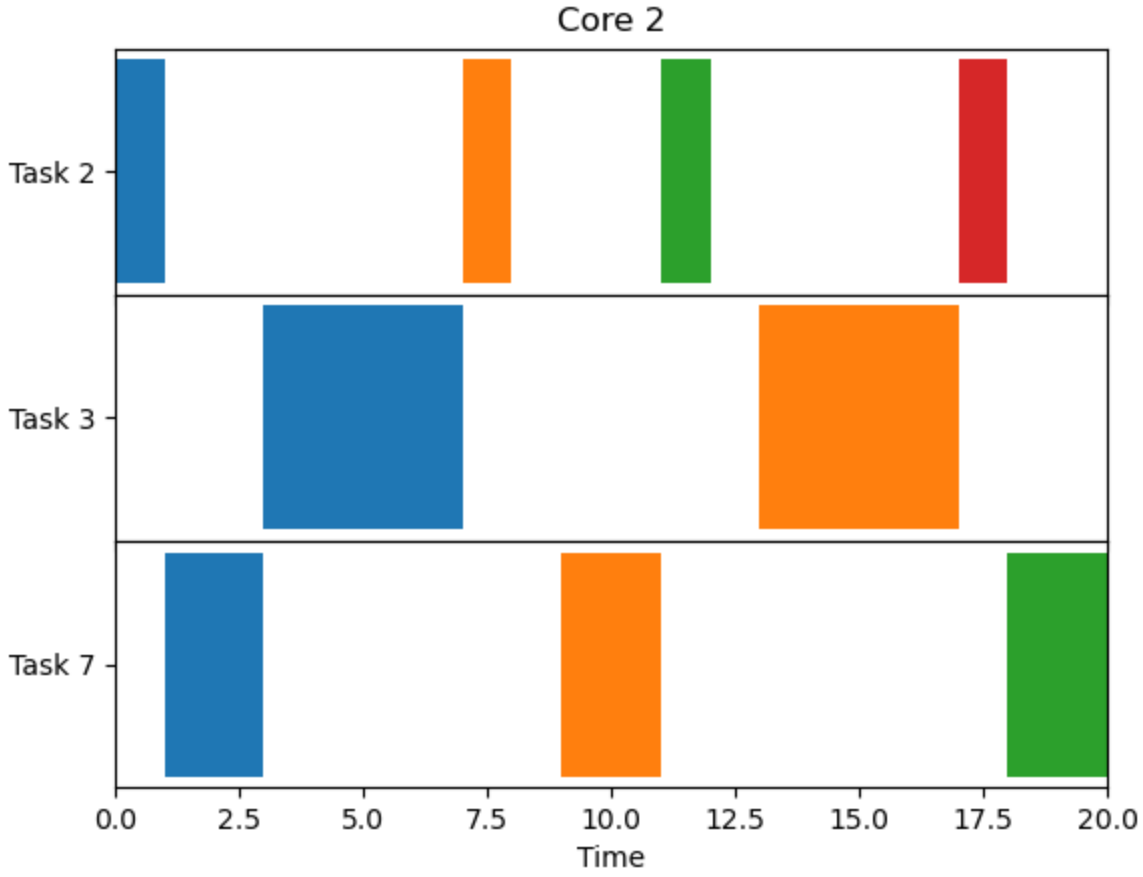


در نمودار بالا، بر حسب تعداد coreهای مختلف fitness را نمایش می‌دهد. همانطور که مشاهده می‌شود، با افزایش تعداد coreها میزان fitness بهتر می‌شود.

در ادامه نمودار اجرای cgo با تابع fitness تلفیقی را نمایش می‌دهیم:



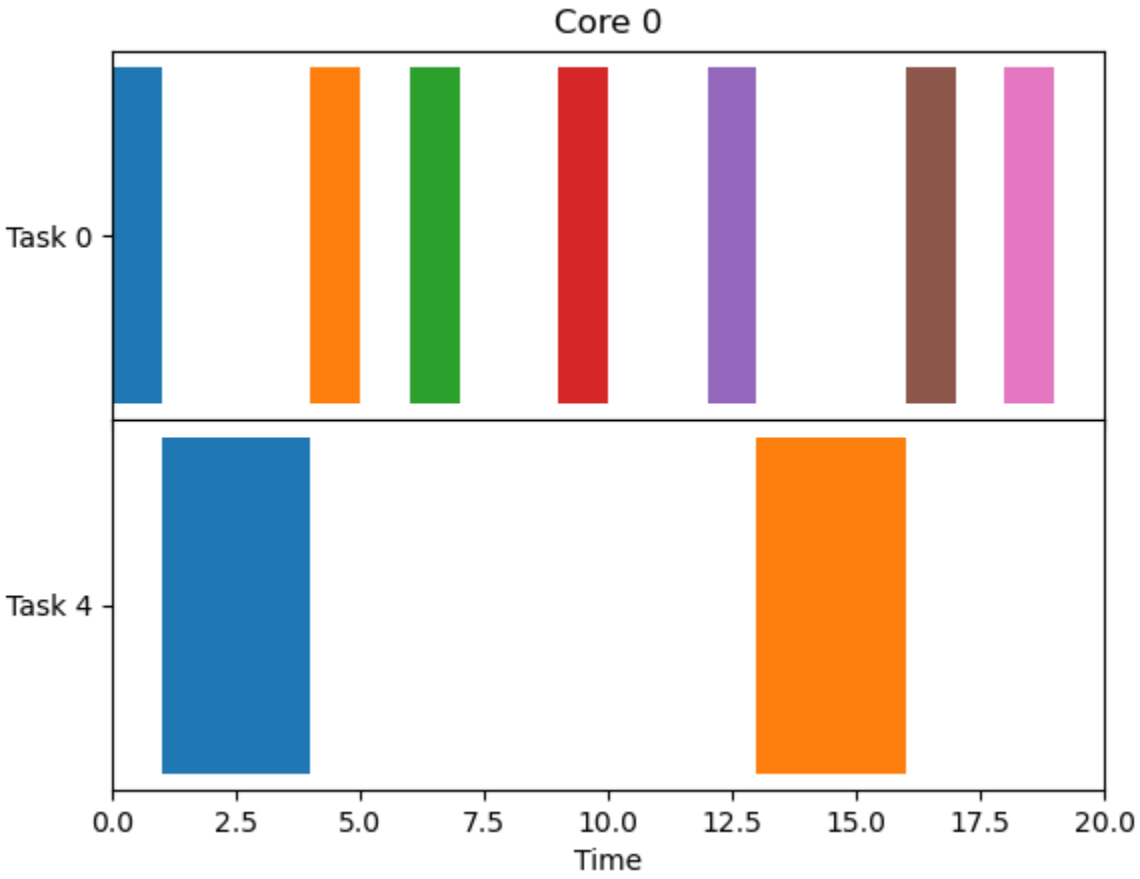

Core 1
Task 1
Task 5
Task 6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Time



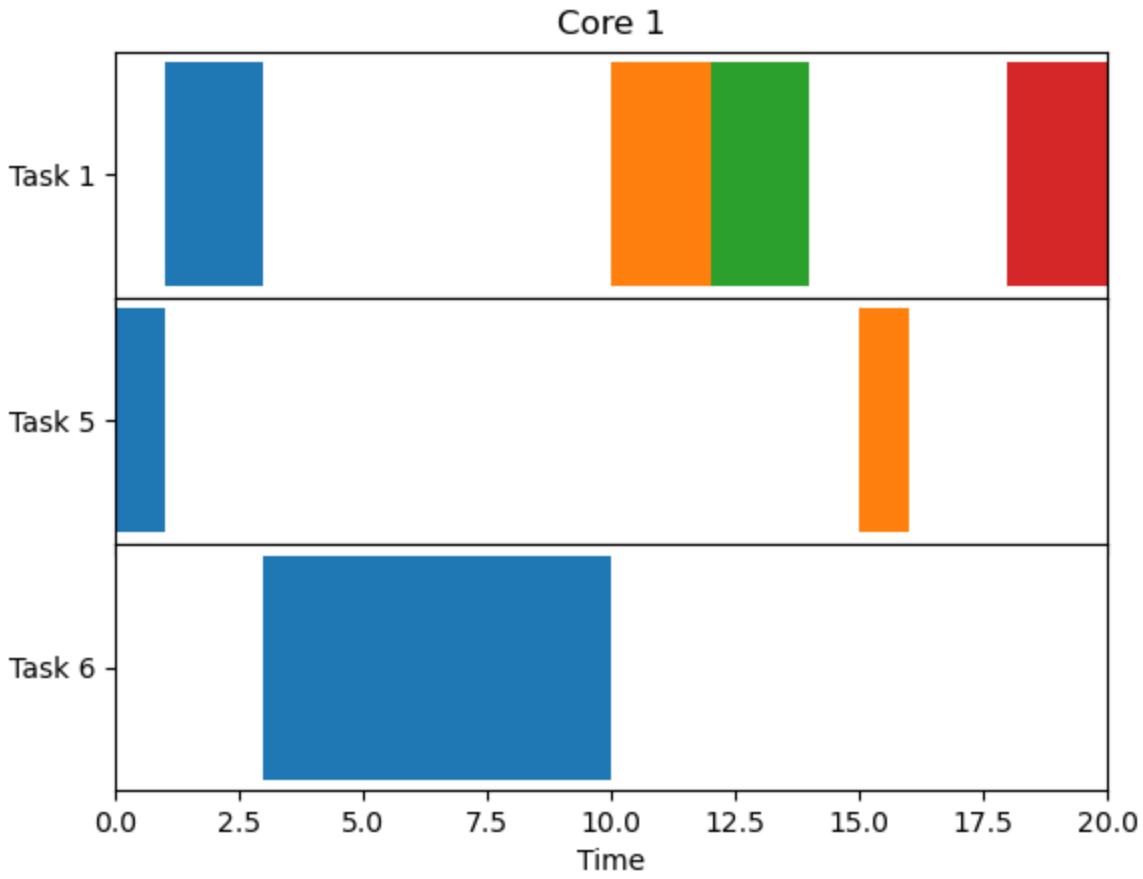
و خروجی json به صورت زیر می‌باشد:

```
{
    "stat": {
        "avg_wait_time": 0.84,
        "sum_wait_time": 21,
        "avg_response_time": 2.76,
        "sum_response_time": 69,
        "avg_delay_time": 0,
        "sum_delay_time": 0,
        "avg_finish_time": 10.6,
        "sum_finish_time": 265,
        "avg_start_time": 8.68,
        "sum_start_time": 217
    },
    "slack": {
        "sum": 11,
        "slack between task 0 and 0": 6,
        "slack between task 5 and 1": 1,
        "slack between task 1 and 5": 2,
        "slack between task 7 and 2": 1,
        "slack between task 3 and 2": 1
    },
    "missed deadlines": 0,
    "algorithm_execution_time": 0.5645513534545898,
    "fitness_values": [
        21,
        33,
        36
    ],
    "fitness_sum": 90,
    "task_mapping": {
        "0": 0,
        "1": 1,
        "2": 2,
        "3": 2,
        "4": 0,
        "5": 1,
        "6": 1,
        "7": 2
    }
}
```

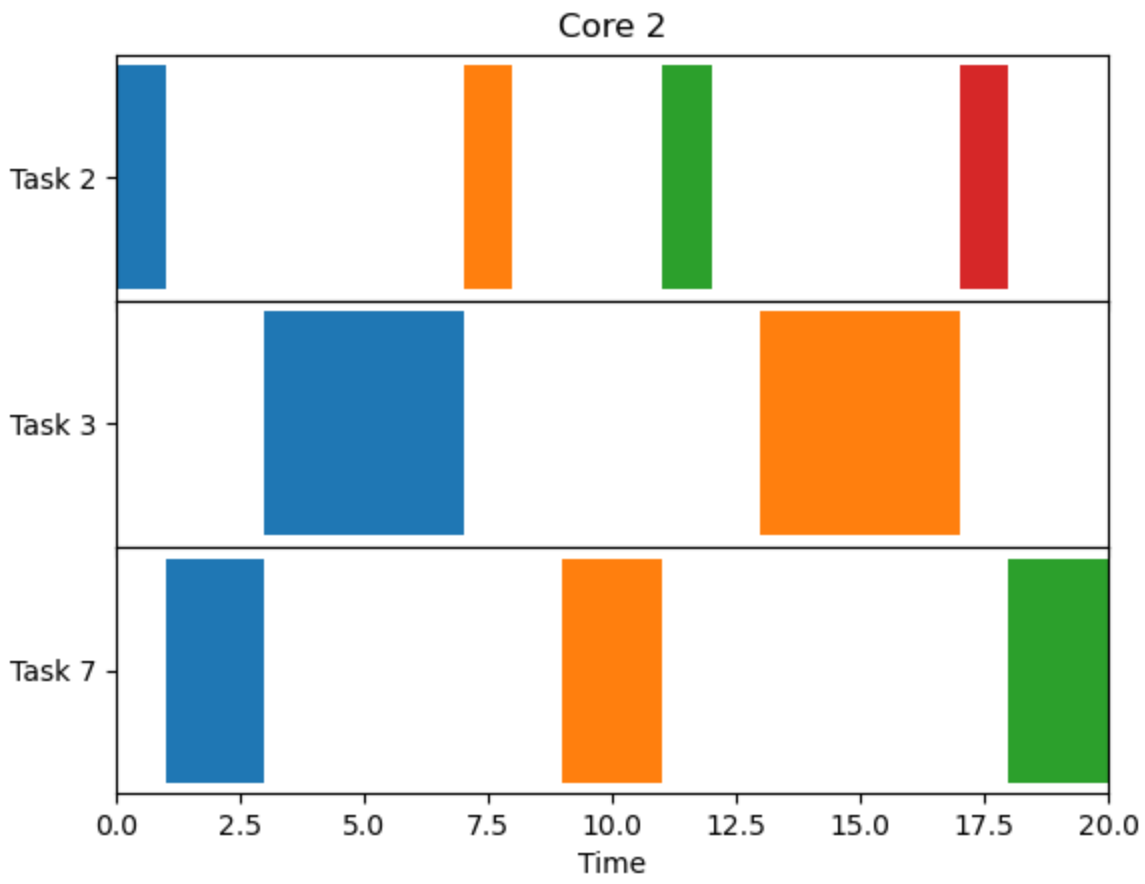
در ادامه نمودار مربوط به اجرای تسک‌ها با الگوریتم AVO می‌آید.



هسته‌ی دوم:



هسته‌ی سوم:


Core 2
Task 2
Task 3
Task 7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Time

خروجی json نیز به صورت زیر می‌باشد:


```
{
    "stat": {
        "avg_wait_time": 0.84,
        "sum_wait_time": 21,
        "avg_response_time": 2.76,
        "sum_response_time": 69,
        "avg_delay_time": 0,
        "sum_delay_time": 0,
        "avg_finish_time": 10.6,
        "sum_finish_time": 265,
        "avg_start_time": 8.68,
        "sum_start_time": 217
    },
    "slack": {
        "sum": 11,
        "slack between task 0 and 0": 6,
        "slack between task 5 and 1": 1,
        "slack between task 1 and 5": 2,
        "slack between task 7 and 2": 1,
        "slack between task 3 and 2": 1
    },
    "missed deadlines": 0,
    "algorithm_execution_time": 0.055519819259643555,
    "fitness_values": [
        21,
        33,
        36
    ],
    "fitness_sum": 90,
    "task_mapping": {
        "0": 0,
        "1": 1,
        "2": 2,
        "3": 2,
        "4": 0,
        "5": 1,
        "6": 1,
        "7": 2
    }
}
```