**گزارش پروژه درس هوش محاسباتی**

**استاد : دکتر مجتبی شاکری**

**موضوع :استفاده از الگوریتم ژنتیک برای حل مسئله P-MEDIAN**


**اعضای گروه:**

**علیرضا دربندی ۹۳۱۲۲۶۸۱۱۶**

**ایمان خالق پرست ۹۳۱۲۲۶۸۱۱۳**

**سحر یوسف نژاد ۹۳۱۲۲۶۸۱۴۰**

**صدف نجفی خواه ۹۳۳۱۲۱۲۰۹۸**


**پاییز ۹۶**

## چکیده

در فصل اول این گزارش شرح مختصری از صورت مسئله و الگوریتم ارائه شده برای حل آن آورده شده است .

در فصل دوم الگوریتم به صورت دقیق بررسی میشود و چالش های آن بررسی میشود و مزایا و معایب آن مورد بررسی قرارگرفته و تغییرات پیشنهادی توسط تیم فنی توضیح داده میگردد.

فصل سوم تست کیس ها و نحوه ی گرفتن ورودی و ارزیابی آن شرح داده میشود و خروجی برنامه تحلیل می گردد.برای درک بهتر،نمودار ها و جداول داده های ورودی و خروجی در نظر گرفته شده است.

فصل آخردر مورد جمع بندی کلی پروژه و داده های تحلیل شده می باشد.

فهرست مطالب

مقدمه ..... ۴

روش شناسی ..... ۵

ارزیابی و نمودار ها ..... ۱۳

نتیجه گیری ..... ۱۵

## فصل اول : مقدمه

مسئله p-median یکی از چالش های مشهور است و برنامه های مختلف در حمل و نقل، توزیع، مکانهای عمومی، انبارها و غیره است. هدف مکان یابی p امکانات محدود برای m سرویس دهنده است. هدف نیز یافتن حداقل مجموع فواصل از نقاط تقاضا تا نقاط تامین امکانات است. مسئله p-median به عنوان یک مسئله np-complete مشهور است.

چندین ایده هیوریستیک منتشر شده است. اما تعداد کمی از راه حل ها در حوزه الگوریتم ژنتیک بوده اند و همچنان این مسئله چالش برانگیز است. در این مقاله یک الگوریتم ژنتیک جدید برای حل مسئله ارائه شده است. الگوریتم پیشنهادی روی چندی نمونه از dataset اجرا و ارزیابی شده است.

هدف مسئله p-median چیست؟

مسئله p-median در اصل یک بهینه سازی گسسته است که هدف آن جاسازی p شماره از امکانات محدود در مختصات است که درخواست های تقاضا را به حداکثر پوشش برساند به نحوی که مجموع فواصل هر درخواست دهنده و امکاناتش به حداقل ممکن برسد

در کنار مسئله p-median، مسئله network یا شبکه وجود دارد که عموما شبیه آن است و برای facility location استفاده دارد. جستجو برای گره های p-median در شبکه یک مسئله مکانی کلاسیک است. درواقع توزیع کالا از انبارهای غیرمتمرکز مفیدتر از انبارهای متمرکز است.

در فصل آینده در مورد روش شناسی حل این الگوریتم توضیح خواهیم داد.

## فصل دوم : روش شناسی

الگوریتمی که برای حل مسئله مطرح شده استفاده کردیم "الگوریتم ژنتیک" است.رویکرد کلی این الگوریتم به شرح زیر است:

- ساختن جمعیت اولیه
- تابع fitness
- مرتب سازی بر اساس fitness ها
- انتخاب و حذف بر اساس fitness ها
- عمل cross over و باز تولید جمعیت
- عمل mutation برای کنترل exploration

در ادامه به شرح هر کدام از موارد فوق خواهیم پرداخت.

### ۱-ساختن جمعیت اولیه:

این جمعیت بصورت تصادفی انتخاب می شود.هر کروموزوم تولیدی که نمایش دهنده ی هر نسل است بصورت آرایه ای یک بعدی از سرویس دهنده ها و سرویس گیرنده ها میباشد.طول این آرایه برابر با اندازه ی سرویس گیرنده ها است یعنی nتا و مقدار هر عضو از این آرایه به شماره ی سرویس دهنده مربوط به آن است که این عدد بین ۰ تا p-1 می باشد.

هر سرویس دهنده یک ظرفیت دارد و تعداد سرویس دهنده ها می تواند کمتر از سرویس گیرنده ها باشد که البته در این الگوریتم طراحی بصورتی بوده که عدم صدق این موضوع خللی در روند محاسبات ایجاد نمی کند.

قطعه کد مربوط به این قسمت که در فایل initial_generation.js میباشد:

```
var generate_random_chromosome = (p, n) => {
  var chromosome = []
  for(var i=0; i<n; i++)
    chromosome[i] = parseInt(Math.random() * p)
  return chromosome
}
```

نمونه ی یک جمعیت تولیدی :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 1 | 6 | 0 | 4 | 1 | 4 |

Service Providers count: 7

Service users Count: 8

**۲-تابع fitness یا برازش:**

دو تابع برای این امر نوشته شده که در کد فعلی تابع اول مورد استفاده قرار گرفته است:

- تابع برازش اول: فرمول آن بصورت زیر است:

$$\left(\frac{\text{worst\_distance} - \text{sum\_of\_distances}}{\text{worst\_distance}}\right) * \left(\frac{\text{total\_needs} - \text{min\_capacity}}{\text{total\_needs}}\right)$$

✓ worst_distance: بدترین فاصله بین سرویس دهنده و گیرنده را محاسبه میکند:

```
var worst_distance = (medians, demands) => {
  var worst = 0
  for(var i=0; i<demands.length; i++){
    var demand = demands[i]
```

```
    var worst_dst_for_current_demand =
the_furthest_median(demand, medians)
    worst += worst_dst_for_current_demand
  }
  return worst
}
var total_needs = (demands) => {
  var total = 0
  for(var i=0; i<demands.length; i++)
    total += demands[i].c
  total_needs_v = total
  return total
}
```

✓ sum_of_distances: مجموع فاصله ی سرویس گیرنده ها و سرویس دهنده ی آن ها را مشخص می کند:

```
var sum_of_distance = (chromosome, medians, demands) =>
{
  var sum = 0
  for(var i=0; i<chromosome.length; i++){
    var demand = demands[i]
    var median = medians[chromosome[i]]
    var current_dst = Math.pow(demand.x - median.x, 2)
+ Math.pow(demand.y - median.y, 2)
    sum += current_dst
  }
  return sum
}
```

✓ total_needs : مجموع نیاز تمام سرویس گیرنده ها

```
var total_needs = (demands) => {
  var total = 0
  for(var i=0; i<demands.length; i++)
    total += demands[i].c
  total_needs_v = total
```

```
  return total
}
```

✓ min _capacity : ظرفیت سرویس دهنده ای که کمترین ظرفیت را دارد.

```
var min_cap_median = (medians) => {
  var min_cap = 999999999
  for(var i=0; i<medians.length; i++)
    if(min_cap > medians[i].c)
      min_cap = medians[i].c

  min_cap_median_v = min_cap
  return min_cap
}
```

- تابع برازش دوم: فرمول آن بصورت زیر است:

$$\frac{\left(\frac{\text{worst\_distance} - \text{sum\_of\_distances}}{\text{worst\_distance}}\right) + \left(\frac{\text{total\_needs} - \text{min\_capacity}}{\text{total\_needs}}\right)}{2}$$

**تفاوت دو تابع:**

اولی ترکیبشان است که احتمال آنهارا در هم تاثیر میدهد اما دومی میانگینشان است که احتمال آن هارا باهم تاثیر می دهد.

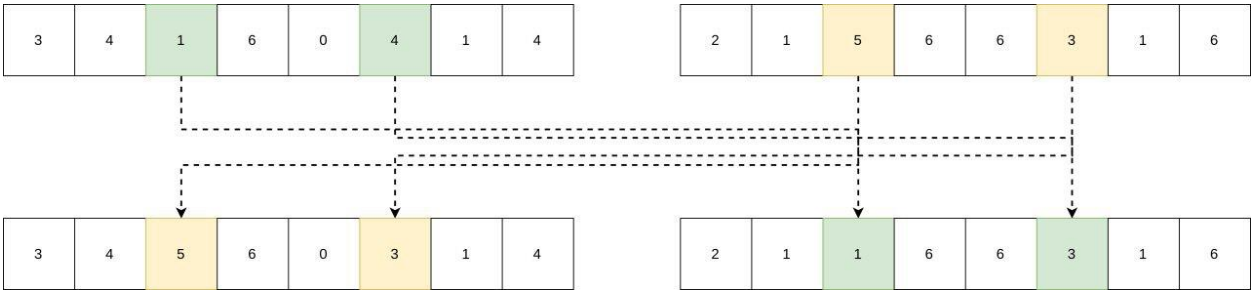
**مراحل دوم و سوم** : انتخاب و حذف بر اساس مقادیر fitness مرتب شده:

این مرحله با هدف حذف فرزندان ضعیف هر نسل انجام می گردد. سیاست کلی بقا بدین شکل از که ابتدا درصدی از فرزندان که کمترین مقدار برازش را دارند انتخاب می شوند

**۴-مرحله عمل cross over و باز تولید جمعیت:**

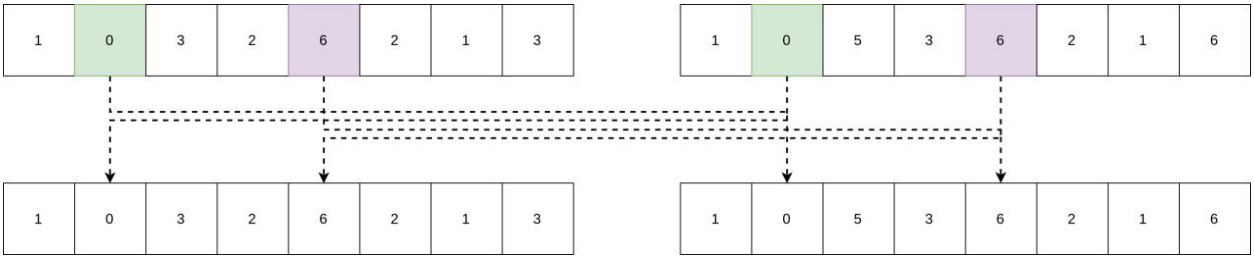
این مرحله با هدف explotation و بازتولید نسل جدیدی است برای جایگزینی فرزندان جدید با فرزندانی که در مرحله ی حذف و بقا از بین رفته اند.برای هر دو تابع برازش cross over پیاده سازی شده است که به شرح زیر است:

ایده مقاله:




```
var crossover_a = (parents, m, n,
crossover_probability) => {
    var childs = [
        [],
        []
    ]
    for (var i = 0; i < n; i++) {
        if (Math.random() >= crossover_probability) {
            childs[0][i] = parents[0][i]
            childs[1][i] = parents[1][i]
        } else {
            // console.log(`swap on ${i}`)
            childs[1][i] = parents[0][i]
            childs[0][i] = parents[1][i]
        }
    }
    return childs
}
```

ایده ما:



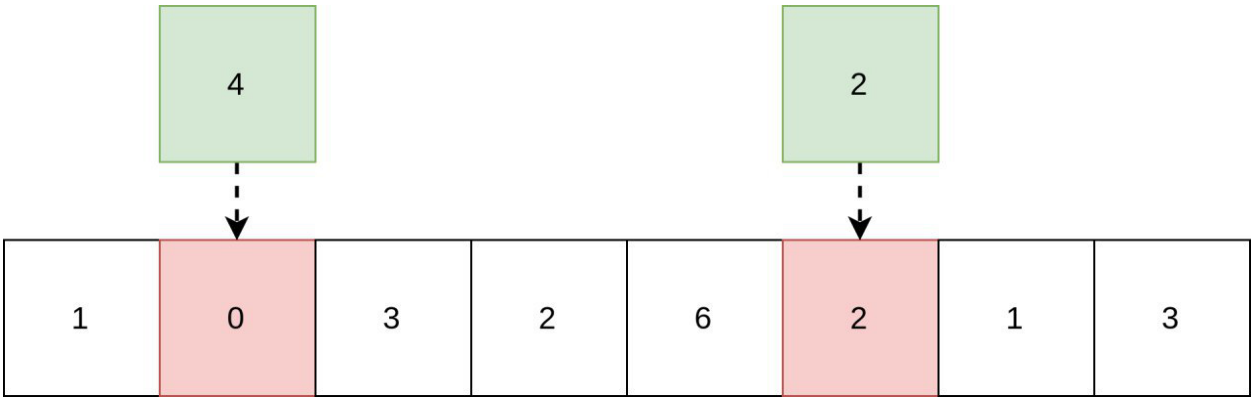

```
var crossover_b = (parents, m, n,
crossover_probability) => {
    for (var i = 0; i < n; i++) {
        var first_index = Math.floor(Math.random() * n)
        var second_index = Math.floor(Math.random() *
n)
            // console.log(`swap on ${first_index} and
${second_index}`)
        var temp = parents[0][first_index]
        parents[0][first_index] =
parents[1][second_index]
        parents[1][second_index] = temp
    }
    return parents
}

module.exports = {
    crossover_a,
    crossover_b
}
```

### ۵- عمل **mutation** برای کنترل **exploration**:

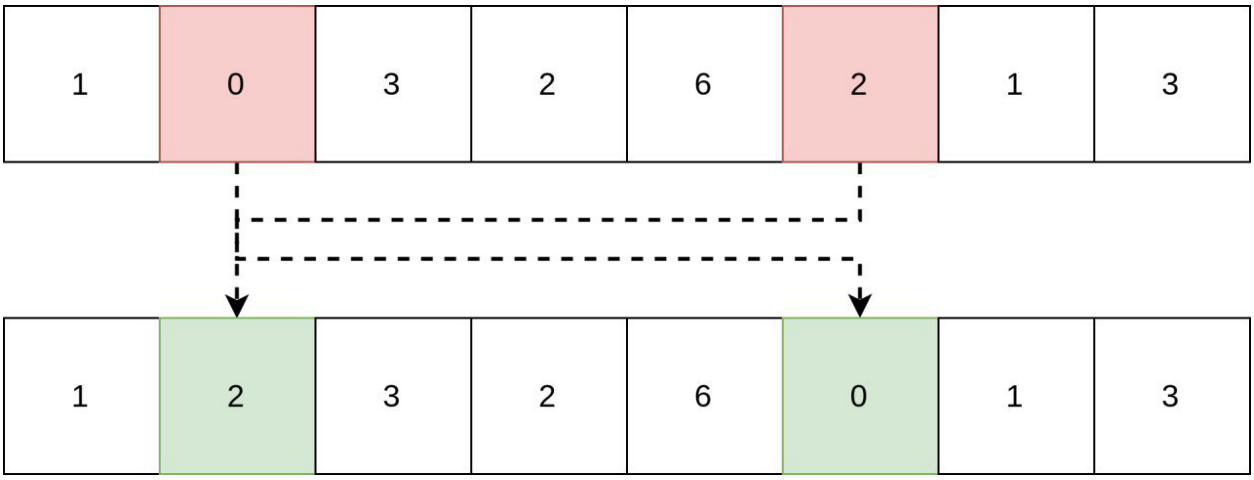
این مرحله با هدف ایجاد مولفه اکتشاف و تولید تنوع بین فرزندان برای رد کردن دره ها است.

ایده ی مقاله:



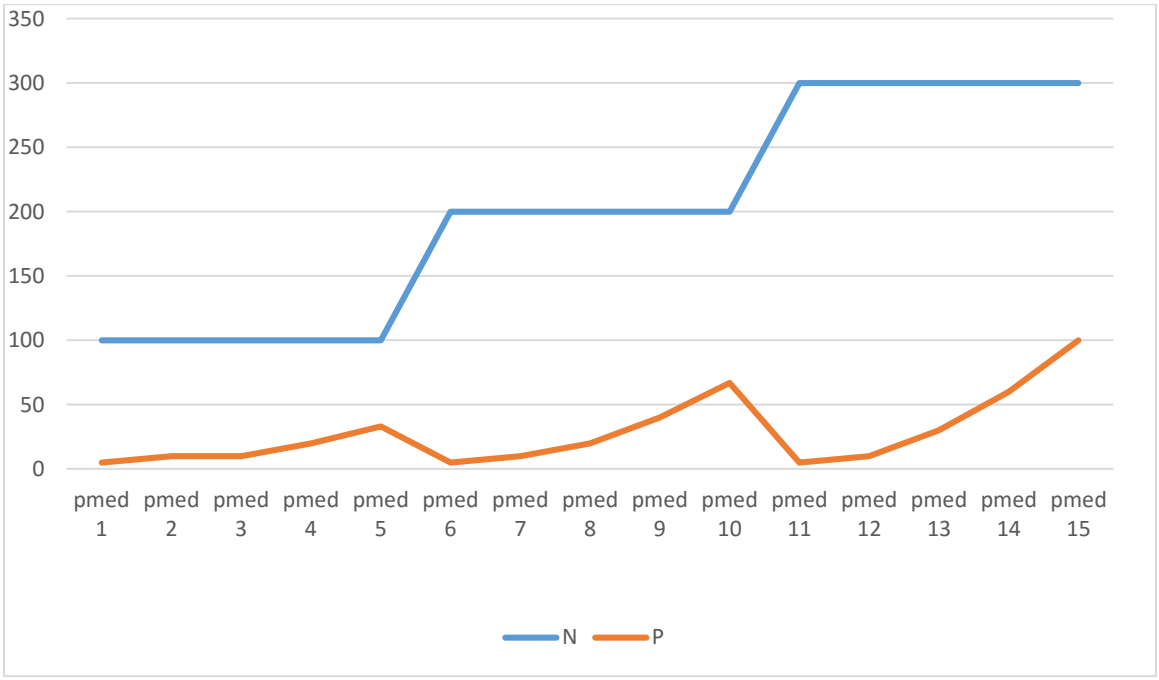

```
var mutation_a = (chromosome, mute_probability, m, n)
=> {
  for(var i=0; i<chromosome.length; i++){
    if(Math.random() < mute_probability){
      // console.log(`mute on gene ${i}`)
      chromosome[i] = Math.floor(Math.random() *m)
    }
  }
  return chromosome
}
```

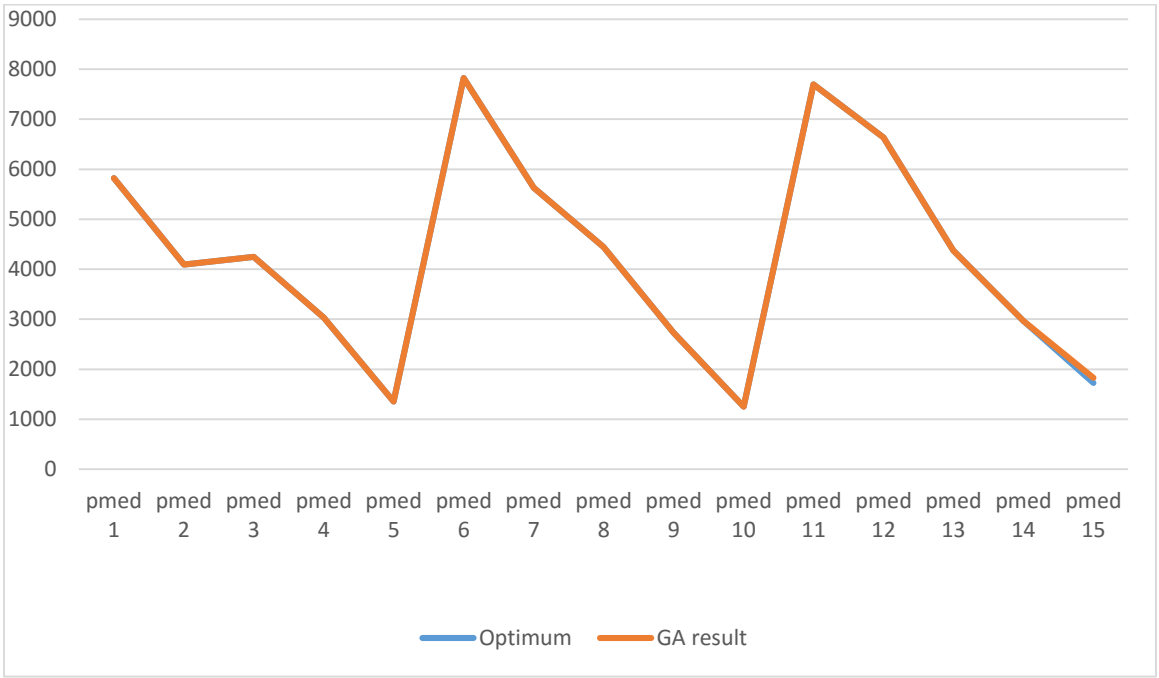
ایده ی ما:

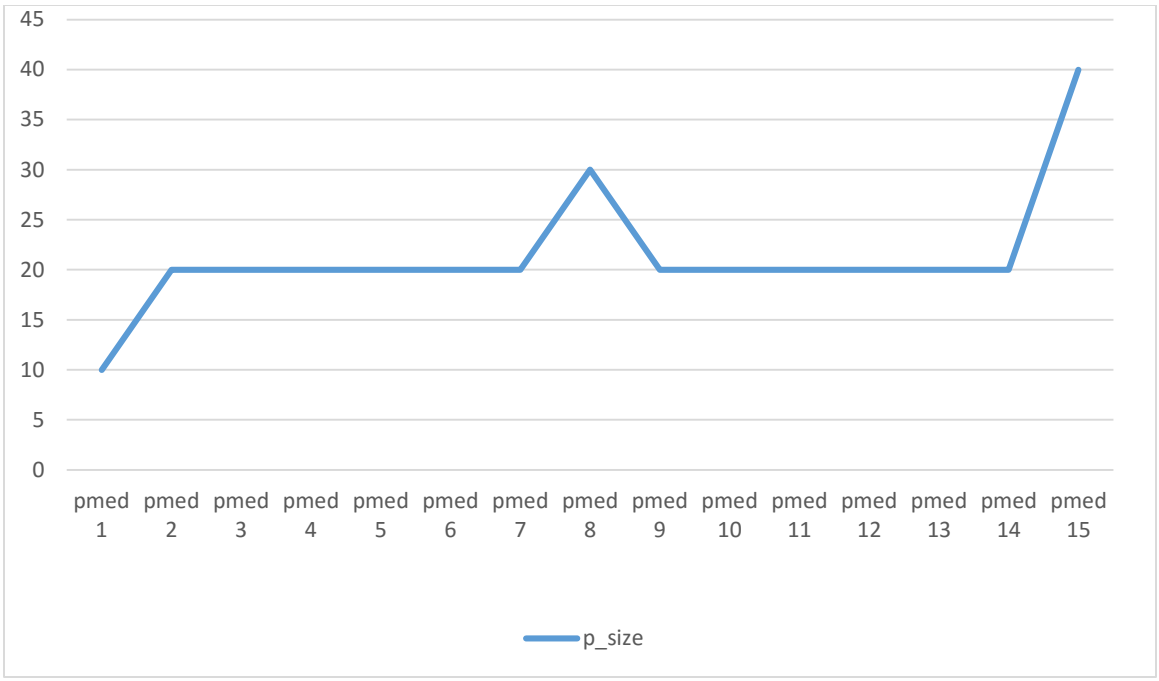


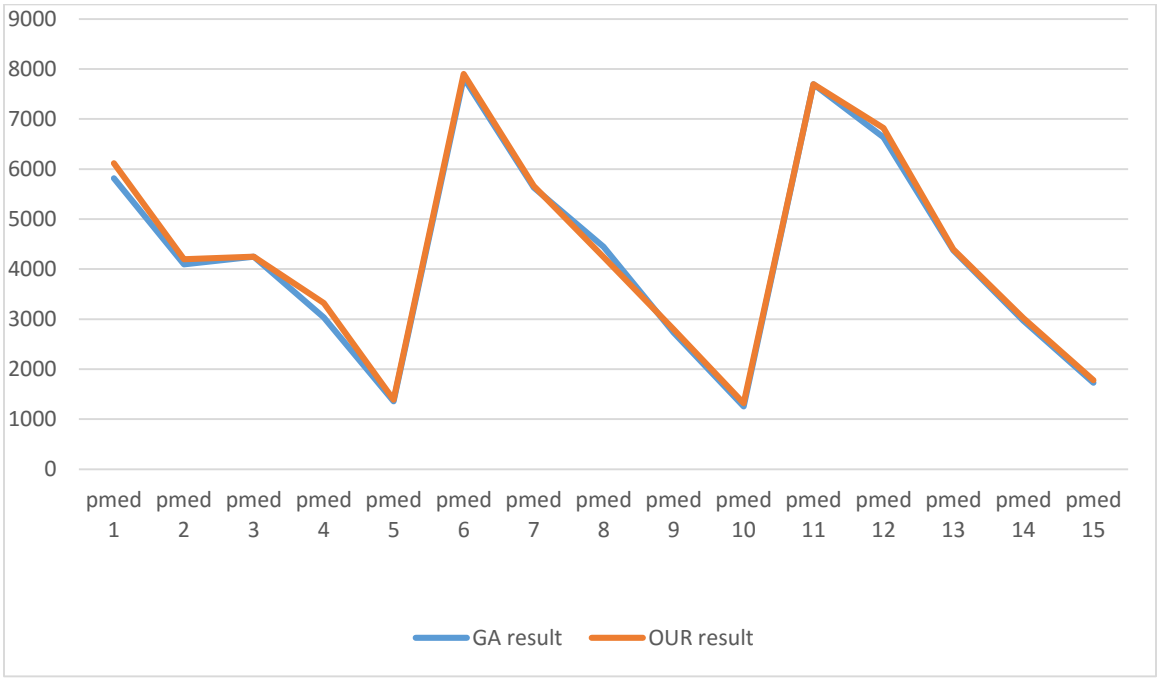

```
var mutation_b = (chromosome, mute_probability, m, n)
=> {
  var swap_counts = Math.floor(mute_probability *n)
  for(var i=0; i<swap_counts; i++){
    var first_index = Math.floor(Math.random() *n)
    var second_index = Math.floor(Math.random() *n)
    // console.log(`swap on ${first_index} and
${second_index}`)
    var temp = chromosome[first_index]
    chromosome[first_index] = chromosome[second_index]
    chromosome[second_index] = temp
  }
  return chromosome
}
```

# فصل سوم : ارزیابی و نمودار ها




45
40
35
30
25
20
15
10
5
0
pmed 1
pmed 2
pmed 3
pmed 4
pmed 5
pmed 6
pmed 7
pmed 8
pmed 9
pmed 10
pmed 11
pmed 12
pmed 13
pmed 14
pmed 15
p_size


9000
8000
7000
6000
5000
4000
3000
2000
1000
0
pmed 1
pmed 2
pmed 3
pmed 4
pmed 5
pmed 6
pmed 7
pmed 8
pmed 9
pmed 10
pmed 11
pmed 12
pmed 13
pmed 14
pmed 15
GA result
OUR result