Verolog Solver Challenge 2017

Martin Matak
Borna Feldsar

Which programming language?

- Very complex problem
- Run it on remote server
- Which server? Which OS is on server?
- Conclusion: Platform independent programming language
- Hence: **Java**

General idea

- Map each request to **some** day
  - create negative requests - days when we pick up tools from customers
- Map each request to **some** vehicle

Two main levels of optimization:

1. Decide on which day request will be executed
2. Optimization of each day:
   i. assign requests to vehicle
   ii. optimize given vehicle route

First approach

First level of optimization:

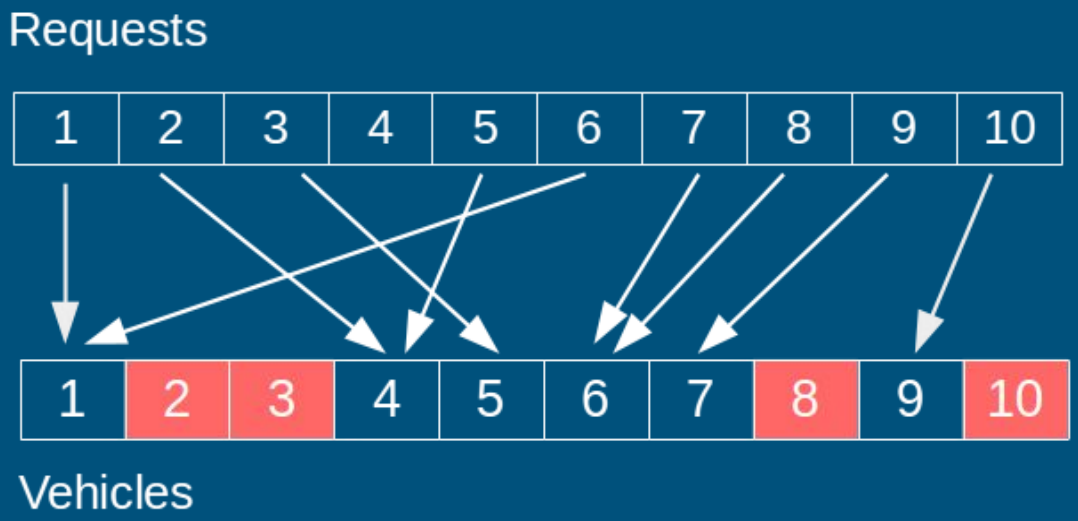
- assign each request to random day in given day window - no optimization

Second level of optimization:

- genetic algorithm

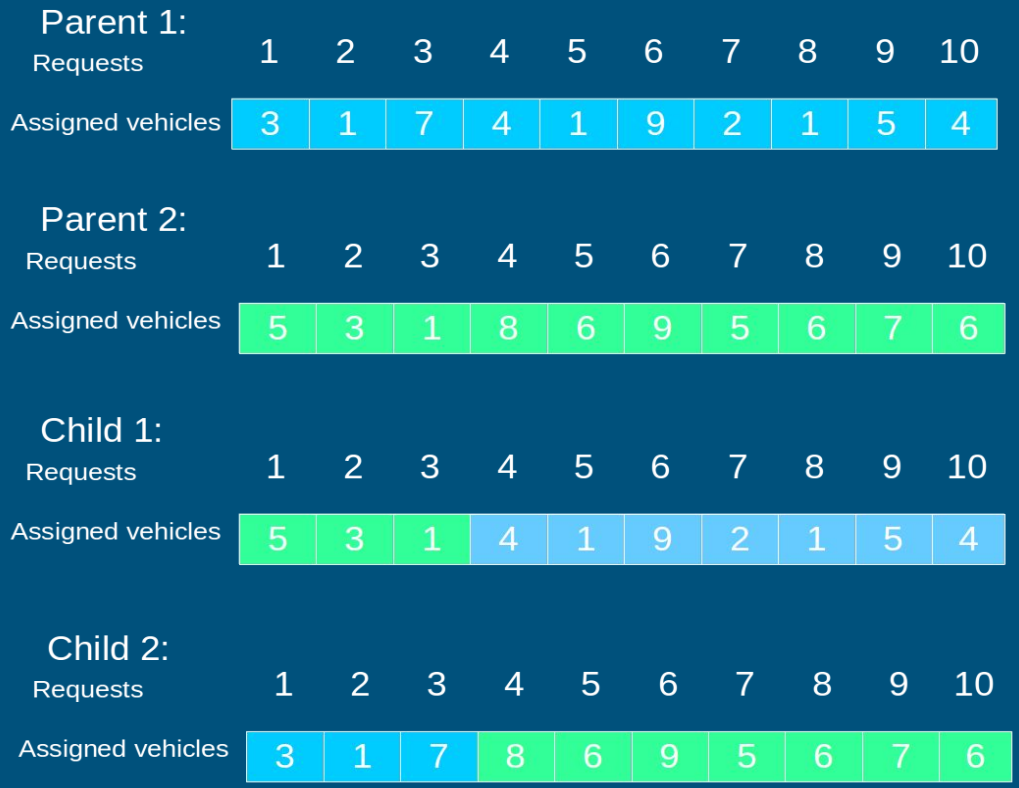
Second Level Genetic Algorithm

- We are looking at requests at the same day
- Chromosome:



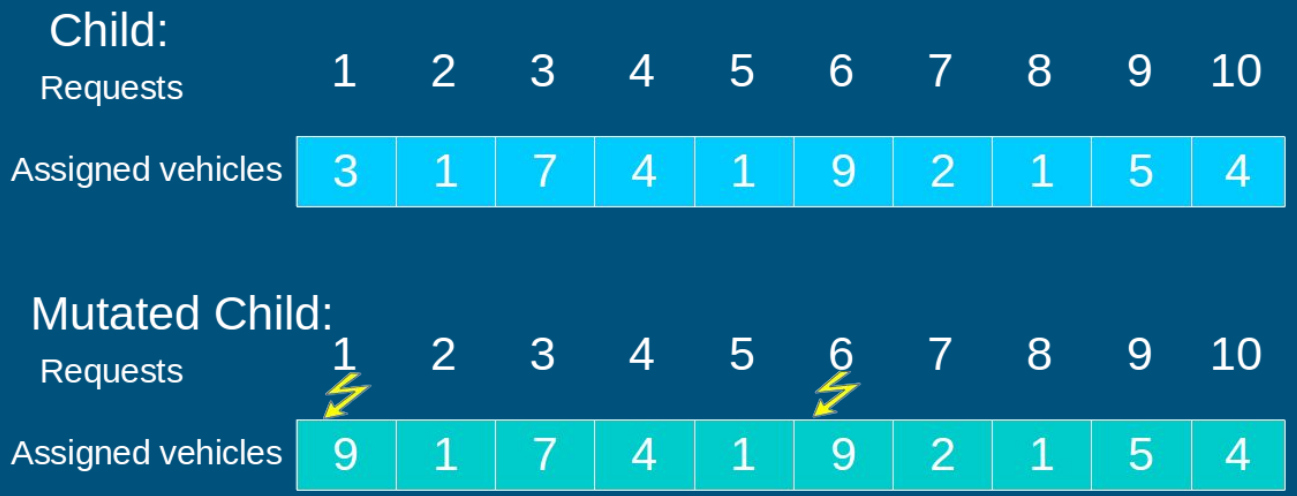
Second Level Genetic Algorithm

- Crossover



Second Level Genetic Algorithm

- Mutation



Second Level Genetic Algorithm

- Evaluation function
  - Since vehicle knows how to optimize its routes he stores exceeded load and vehicles distance while optimizing
  - Incremental evaluation function in a way that vehicle optimizes only days where new request is added or some request is removed
  - Total cost of a solution. Punishment is calculated based on exceeded distance and exceeded capacity

… but it was so hard to optimize routes for vehicles when mapping from requests to days was random

problem: more often than not, we had capacity overload and tool usage overload

Second approach

Map each request to the *most empty* day for that tool which request needs.

We were greedy …

… so our results were once again not feasible, but not that often anymore

Third approach

We make map <Day, <tool_id, #usage>>

We store how many tools of specific id are used on which day

idea: DFS with pruning

Node: request.id, picked day for that request

children: next request, all possible days for that next request

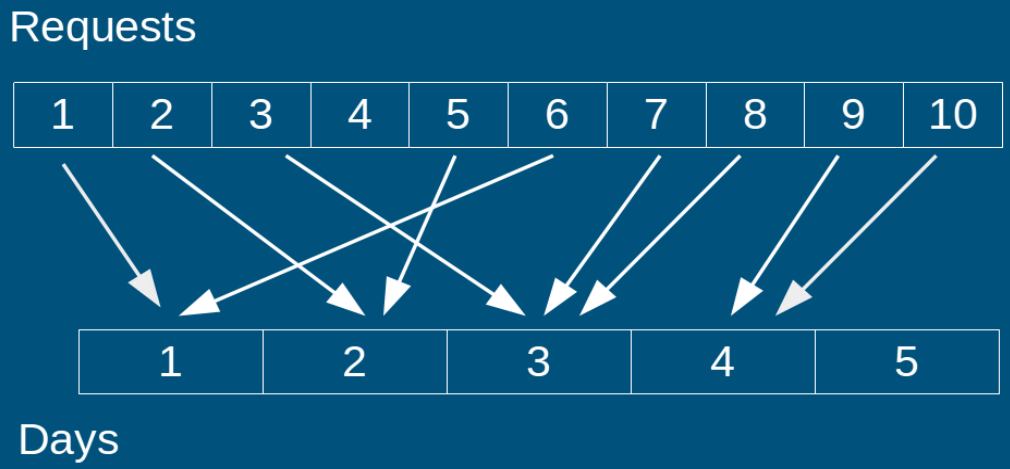
Requests are sorted: tool_id, length of window frame, number of tools, but…

very time consuming with large instances!

Fourth approach

So why not use GA for deciding days also? :)

- Our Chromosome
- We are looking at all instance requests



Fourth approach

- Two types of crossover. Requests which are involved in crossover are :

1. all chromosome requests
2. only requests that are linked to day tool slot where tool usage that day is exceeded. Day tool slot is chosen using wheel selection regarding exceeded usage

The second one is chosen with probability of 0.6 and only if there are exceeded slots

Fourth approach

- Mutation
- Criteria of choosing requests is the same as in crossing

Child:

Requests	1	2	3	4	5	6	7	8	9	10
Assigned days	3	1	2	4	1	1	2	1	4	4

Mutated Child:

Requests	1	2	3	4	5	6	7	8	9	10
Assigned days	3	1	4	4	1	1	2	4	4	4

Fourth approach

- Evaluation function
  - Each Chromosome stores map <Day, <tool_id, #usage>>
  - After request is assigned to some day or detached map is updated only where change is made
  - Chromosome has sorted Set of exceeded slots (Day, toolId)
  - Maximal number of tools consumption is used as cost and as punishment exceeded value of each slot

Further improvements

- improve strategy for binding from request to day (1st phase)
  - take into account edge cases about usage of tool
- improve algorithm for route optimization on one vehicle (2nd phase)
  - so far only hill climbing
- try with some other crossovers and mutations in 2nd phase
  - crossover with 2 or three points of intesection
  - mutation could take into account already provided route for some vehicle

you want to give it a shot? pull request at:
https://github.com/Cibale/VerologSolverChallenge2017

Source code

```
public void decideDays() {
    // random method just to start with
    for (int i = 0; i < model.requests.length; i++) {
        Request request = model.requests[i];
        request.pickedDayForDelivery = request.firstDayForDelivery +
                rand.nextInt(request.lastDayForDelivery + 1 - request.firstDayForDelivery);
    }
    createNegativeRequests();
}
```

Source code

```
public void decideDaysGreedy() {
    Map<Integer, Map<Integer, Integer>> days = new HashMap<>();
    for (Request request : model.requests) {
        //for every request look in days how many tools of that ids is in use that day
        int minNumOfToolsDay = -1;
        int minDay = -1;
        // find minDay - day with minimum tools of that kind
        for (int i = request.firstDayForDelivery; i <= request.lastDayForDelivery; i++) {
            //map (toolId, numOfTool)
            Map<Integer, Integer> day = days.get(i);
            if (day == null) {
                day = new HashMap<>();
                days.put(i, day);
            }
            Integer numOfTools = day.getOrDefault(request.toolId, 0);
            if (minNumOfToolsDay == -1 || minNumOfToolsDay > numOfTools) {
                minNumOfToolsDay = numOfTools;
                minDay = i;
            }
        }
        request.pickedDayForDelivery = minDay;
        for (int j = request.pickedDayForDelivery; j < request.pickedDayForDelivery + request.durationInDays; j++) {
            Map<Integer, Integer> day = days.get(j);
            if (day == null) {
                day = new HashMap<>();
                days.put(j, day);
            }
            Integer numOfTools = day.getOrDefault(request.toolId, 0);
            numOfTools += request.numOfTools;
            day.put(request.toolId, numOfTools);
        }

    }
    createNegativeRequests();
}
```

Source code

```
public void decideDaysDFS() {
    DFS dfs = new DFS(this, Arrays.asList(model.requests));
    long tStart = System.currentTimeMillis();
    List<Request> requests = dfs.run();
    long tEnd = System.currentTimeMillis();
    long tDelta = tEnd - tStart;
    double elapsedMinutes = tDelta / 1000.0 / 60;
    System.out.println("Minutes needed: " + elapsedMinutes);
    requests.sort(Comparator.comparingInt(o -> o.id));
    for (int i = 0; i < model.requests.length; i++) {
        model.requests[i] = new Request(requests.get(i));
    }
    try {
        FileOutputStream out = new FileOutputStream("secondInstance.out");
        ObjectOutputStream oos = new ObjectOutputStream(out);
        oos.writeObject(requests);
        oos.flush();
    } catch (Exception e) {
        System.out.println("Problem serializing: " + e);
    }
    createNegativeRequests();
}
```

Source code

```
public DaysChromosome[] crossParents(DaysChromosome parent1, DaysChromosome parent2, double CROSSOVER_PROBABILITY) {
    if (Math.random() > CROSSOVER_PROBABILITY) {
        return new DaysChromosome[]{new DaysChromosome(parent1), new DaysChromosome(parent2)};
    }
    DaysChromosome child1 = new DaysChromosome(parent1);
    DaysChromosome child2 = new DaysChromosome(parent2);

    crossChild(child1,parent1,parent2);
    crossChild(child2,parent2,parent1);

    return new DaysChromosome[]{child1, child2};
}

private void crossChild(DaysChromosome child1, DaysChromosome parent1, DaysChromosome parent2){
    double probability = 0.6;
    if(Math.random() > probability && parent1.sumExceeded != 0) {
        DaysChromosome.ExceededToolId exceededToolId1 = parent1.proportionalySelectExceedToolId();
        List<Request> linkedRequests1 = child1.daysMap.get(exceededToolId1.day).get(exceededToolId1.toolId).requestList;
        int splitted = ThreadLocalRandom.current().nextInt(linkedRequests1.size());
        // splitted = linkedRequests1.size();
        for (int i = 0; i < splitted; i++) {
            child1.detachRequest(child1.requests[i]);
            child1.assignRequestToDay(child1.requests[i], parent2.requests[i].pickedDayForDelivery);
        }
    }else {
        int splitPoint = ThreadLocalRandom.current().nextInt(parent1.requests.length);
        for (int i = 0; i < splitPoint; i++) {
            child1.detachRequest(child1.requests[i]);
            child1.assignRequestToDay(child1.requests[i], parent2.requests[i].pickedDayForDelivery);
        }

    }
}
```

Source code

```
public VehicleChromosome[] crossParents(VehicleChromosome parent1, VehicleChromosome parent2, double CROSSOVER_PROBABILITY) {
    if (Math.random() > CROSSOVER_PROBABILITY) {
        return new VehicleChromosome[]{new VehicleChromosome(parent1), new VehicleChromosome(parent2)};
    }
    int splitPoint = ThreadLocalRandom.current().nextInt(parent1.requests.length);
    VehicleChromosome child1 = new VehicleChromosome(parent1);
    VehicleChromosome child2 = new VehicleChromosome(parent2);
    for (int i = 0; i < splitPoint; i++) {

        child1.vehicles[child1.requests[i].correspondingVehicleId].removeRequest(i);
        child1.requests[i].correspondingVehicleId = parent2.requests[i].correspondingVehicleId;
        child1.vehicles[child1.requests[i].correspondingVehicleId].addRequest(i);


        child2.vehicles[child2.requests[i].correspondingVehicleId].removeRequest(i);
        child2.requests[i].correspondingVehicleId = parent1.requests[i].correspondingVehicleId;
        child2.vehicles[child2.requests[i].correspondingVehicleId].addRequest(i);
    }
    for (int i = 0; i < parent1.vehicles.length; i++) {
        child1.vehicles[i].update();
        child2.vehicles[i].update();
    }

    return new VehicleChromosome[]{child1, child2};
}
```

Source code

```
public void addRequest(Integer newRequestId) {
    Request newRequest = vehicleChromosome.requests[newRequestId];

    DayRoute dayRoute = dayRouteMap.get(newRequest.pickedDayForDelivery);
    if (dayRoute == null) {
        dayRoute = new DayRoute(vehicleChromosome.model);
        dayRouteMap.put(newRequest.pickedDayForDelivery, dayRoute);
    }
    changedDays.add(newRequest.pickedDayForDelivery);
    dayRoute.add(newRequest);
    requestListIds.add(newRequestId);
}

public void removeRequest(Integer requestId) {
    Request request = vehicleChromosome.requests[requestId];
    DayRoute dayRoute = dayRouteMap.get(request.pickedDayForDelivery);
    changedDays.add(request.pickedDayForDelivery);
    dayRoute.remove(request);
    requestListIds.remove(requestId);
}
```

Source code

```
public void assignRequestToDay(Request request, Integer pickedDay) {
    for (int i = pickedDay; i <= pickedDay + request.durationInDays; i++) {
        Map<Integer, UsageToolRequests> toolUsageMap = this.daysMap.get(i);
        if (toolUsageMap == null) {
            toolUsageMap = new HashMap<>();
            daysMap.put(i, toolUsageMap);
        }
        UsageToolRequests usageToolRequests = toolUsageMap.getOrDefault(request.toolId, new UsageToolRequests());
        usageToolRequests.usage += request.numOfTools;
        usageToolRequests.requestList.add(request);
        toolUsageMap.put(request.toolId, usageToolRequests);

    }
    request.pickedDayForDelivery = pickedDay;

}

public void detachRequest(Request request) {
    for (int i = request.pickedDayForDelivery; i <= request.pickedDayForDelivery + request.durationInDays; i++) {
        Map<Integer, UsageToolRequests> toolUsageMap = this.daysMap.get(i);
        UsageToolRequests usageToolRequests = toolUsageMap.get(request.toolId);

        usageToolRequests.usage -= request.numOfTools;
        usageToolRequests.requestList.remove(request);
        toolUsageMap.put(request.toolId, usageToolRequests);
    }
    request.pickedDayForDelivery = -1;
}
```

Source code

```
public ExceededToolId proportionalySelectExceedToolId() {
    if(sumExceeded == 0){
    }
    int rand = ThreadLocalRandom.current().nextInt(sumExceeded);
    int counter = 0;
    ExceededToolId tmp = null;
    for (ExceededToolId exceededToolId : exceededToolIdSet) {
        counter += exceededToolId.execeeded;
        if (rand <= counter) {
            tmp = exceededToolId;
            break;
        }
    }


    return tmp;
}
```

Results

Assigning requests to random days

Test instance	cost
1	Tool usage exceeded
2	10 708 394
3	Vehicle max distance exceeded
4	Tool usage exceeded
5	9 974 575
6	7 279 670
7	Vehicle max distance exceeded
8	Not feasible solution
9	9 868 304 245
10	10 429 822 495

Results

Assigning requests to days using DFS

Test instance	cost
1	Tool usage exceeded
2	9 464 533
3	Vehicle max distance exceeded
4	Tool usage exceeded
5	10 033 276
6	863 048 961
7	9 767 415 522
8	1 456 129 919
9	9 844 465 096
10	10 442 946 834

Results

Assigning requests to days using Genetic algorithm

Test instance	cost
1	Tool usage exceeded
2	9 127 248
3	Vehicle max distance exceeded
4	Tool usage exceeded
5	10 027 935
6	853 567 624
7	Vehicle max distance exceeded
8	Vehicle max distance exceeded
9	9 862 246 397
10	10 402 509 848

Lessons learned

- First work with softened instance of problem
- Approach to these kind of problems:
  - 1. Make it work anyhow
  - 2. Optimize
- Testing of code is VERY useful
- We got a feeling about parameters for genetic algorithm
- It is really hard to organize how and where vehicles should travel :)

Thank you for your patience!

.. and we hope we will see each other in the finals of next Verolog Solver challenge :)