FACULTEIT WETENSCHAPPEN

Dynamisch Vehicle routing problem met heterogene vloot van auto's en drones

Glenn Feys
Studentennummer: 01703744

Promotor: prof. dr. Veerle Fack

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de informatica

Academiejaar 2021-2022

UNIVERSITEIT GENT

Samenvatting

In deze masterproef onderzoeken we hoe de toekomst van onze pakjessector er kan uitzien en hoe drones daarin een grote rol spelen. In 2013 onthulde het Amerikaanse e-commercebedrijf Amazon dat het onderzoek ging voeren naar drone delivery. Zo zouden in de toekomst leveringen mogelijk zijn binnen de 30 minuten door gebruik te maken van drones. De multinational liet weten dat deze toekomst dichterbij kan zijn dan we denken, wat aanleiding gaf voor andere onderzoeken om zich bezig te houden met drone delivery. Dat was in het bijzonder het geval nadat de eerste paper verscheen over drone delivery, namelijk die van Murray & Chu [48] over the flying sidekick traveling salesman problem (FSTSP) & parallel drone scheduling traveling salesman problem (PDSTSP). De doelstelling van deze scriptie is een algoritme te vinden waarmee we een planning kunnen maken voor n-wagens en m-drones die op een zo efficiënt mogelijke manier van een depot naar een aantal opgegeven locaties kunnen gaan. Daarbij wordt er geprobeerd om de totale tijd van de operaties (de makespan) zo kort mogelijk te houden. Wagens kunnen op één route naar verschillende locaties gaan, terwijl drones slechts één locatie per trip kunnen bezoeken, waarna ze moeten terugkeren naar het depot.

Om het PDSVRP op te lossen, wordt er een beroep gedaan op een hybrid genetic search algoritme (HGS) van Vidal et al. [66]. HGS is een genetisch algoritme met geavanceerde population management en diversity management. Dit onderzoek breidt dat algoritme uit door ook te werken met drones, dit algoritme noemen we het hybrid genetic search with drones (HGSD). Het algoritme laat toe om een populatie op te bouwen die diverse oplossingen heeft, zodat we ver verschillende paden kunnen verder zoeken. Om de oplossingsruimte te doorzoeken, wordt gebruik gemaakt van een combinatie van slimme mutaties en cross-overs. Er werden heuristieken gedefinieerd zodat de onnodige onderdelen in de oplossingsruimte gesnoeid worden. Daarvoor worden niet alleen enkele bekende mutaties zoals swap, relocate en 2-OPT toegepast, er worden ook enkele nieuwe mutaties gemaakt specifiek om het vooropgestelde probleem aan te pakken. Voor elke mutatie wordt er een heuristiek geformuleerd.

We gebruiken een biased fitness-functie die niet enkel rekening houdt met de score van de oplossing, maar ook met de diversiteit die deze oplossing toevoegt aan de populatie. Oplossingen met de beste biased fitness-score laten we doorgaan naar de volgende generatie. Door de veelvuldigheid aan mutaties komen uiteindelijk behoorlijke oplossingen naar voren. We voegen regelmatig ook random restarts toe om de diversiteit van de populatie in stand te houden. We doen dat in een ketting-hiërarchische structuur, zoals te zien is op afbeelding 4.4. Er wordt voor deze deze structuur gekozen, zodat we efficiënt kunnen blijven zoeken en levels kunnen toevoegen.

Bovendien bekijken we ook welke aanpassingen bevorderlijk zijn, zodat HGSD een online algoritme wordt dat in staat is te functioneren in een setting van same-day delivery: een systeem waarin pakjes de dag van de bestelling nog geleverd worden. We veronderstellen namelijk dat drones hoofdzakelijk gebruikt zullen worden om dat systeem mogelijk te maken. Terwijl het algoritme loopt, moeten er op elk moment locaties kunnen bijkomen en weggaan en moet het mogelijk zijn dat bepaalde routes verwijderd worden uit de berekening wanneer deze gereden worden. Het algoritme was ontwikkeld met dit scenario in het achterhoofd, waardoor we enkel noodzakelijke aanpassingen dienden door te voeren.

We stellen de huidige testmethode van PDSTSP/PDSVRP ter discussie en ontwikkelen zelf een test- en evaluatiemethode om PDSVRP-oplossingen objectief te evalueren. Onze resultaten worden ook vergeleken met de beste oplossing die onderzoeken rond PDSVRP/PDSmTSP en PDSTSP [55][58][17][50][46] naar voren brengen. Daaruit blijkt dat de oplossingen uit deze scriptie opmerkelijke verbeteringen aanbrengen dan de huidige beste oplossingen. Onze large-scale instanties zijn significant beter, we publiceren voor het eerst oplossingen van PDSVRP voor instanties met 250-11.000 locaties en we maken een open-source implementatie van HGSD openbaar via Github.

A hybrid genetic search algorithm for the parallel drone scheduling vehicle routing problem

Glenn Feys Veerle Fack

Abstract

In this paper we take a look at drone delivery, and how this can potentially change the way last-mile delivery is done today. We therefore use the parallel drone scheduling vehicle routing problem, which was first introduced by (Murray & Chu, 2015). We discuss our solution of this problem that uses a hybrid genetic search approach that was inspired by the work of (Vidal et al., 2012) and updated it to be able to work with drones and handle our new constraints. We then discuss the use-case of drones in same-day delivery and extend our algorithm to be able to work online in this kind of environment. Finally, we test our solution against the current state-of-the-art, where we were able to make big improvements on large instances. We also critique the current test-/evaluation methods and produce our own. We also provide an open-source implementation of the algorithm on Github.

1. Introduction

In 2013 Amazon came with the announcement of their research on prime air (Amazon), a revolutionary concept of drone delivery. In the last couple of years drones and other unmanned vehicles have seen a significant increase in applications and research. They are currently frequently used in the film-, surveillance-, and maintenance industry to name a few. But drone delivery could well be the next big revolution in the transportation sector and change the way we receive parcels. It can potentially lower costs, reduce delivery times and make the deliver process more sustainable for the environment. Research regarding the technical side of drone delivery is already well-established, but research regarding algorithms for drone delivery has only really started since the paper of Murray and Chu (Murray & Chu, 2015) on the flying sidekick traveling salesman problem (FSTSP). This was the first research done about adding unmanned aerial vehicles to the TSP/VRP. In their paper they have formally defined two variants of the problem. The first is called Flying sidekick TSP (FSTSP), where a truck collaborates with a drone in tandem, and the drone can take-off and land of the truck. This way both the drone and truck deliver packages, but they work together in a tandem. The second variant is called the parallel drone scheduling TSP (PDSTSP), here the truck and drone start from the same depot, but the drone delivers packages independently from the truck. The drone just goes back and forth between customers and the depot to deliver the packages. The ideas have later been generalized with m trucks and n drones into a vehicle routing problem with drones. The problem we will be focusing on in this paper is the VRP variant of the PDSTSP, the parallel drone scheduling vehicle routing problem (PDSVRP). We used a hybrid genetic search (HGS) algorithm inspired by (Vidal et al., 2012) and extended it to be able to handle drones and called it hybrid genetic search with drones (HGSD). Since drones will be used to reduce delivery times and go to a system that can deliver the same day of ordering, we also discuss how we can extend the HGSD to a dynamic, online model where orders can be added, and routes can be driven at any time in the calculations. The focus of the paper is on large scale instances that reflect real-life scenarios. The HGSD algorithm has also proven to be competitive against the current state-of-the-art beating a lot of best-known solutions, especially with a big leap for large instances. We also released an open-source implementation on Github of HGSD and produced a new method to test and evaluate PDSVRP solutions.

2. Parallel drone scheduling vehicle routing problem (PDSVRP)

The vehicle routing problem (VRP) was first introduced by (Dantzig & Ramser, 1959) and was a generalization of the traveling salesman problem (TSP) with n trucks instead of one. The capacitated VRP is defined by a fleet of trucks, a depot where trucks start from and a set of locations the trucks need to visit. Each location must be visited by exactly one truck. Each location has a certain demanded capacity and vehicles also have a maximum capacity. The goal is to construct routes that minimize the total distance traveled by the fleet and does not violate the capacity constraint of any truck. Since the TSP is NP-hard and the VRP is a generalization on this problem, the VRP is also NP-hard. The parallel drone scheduling vehicle routing problem (PDSVRP) is an

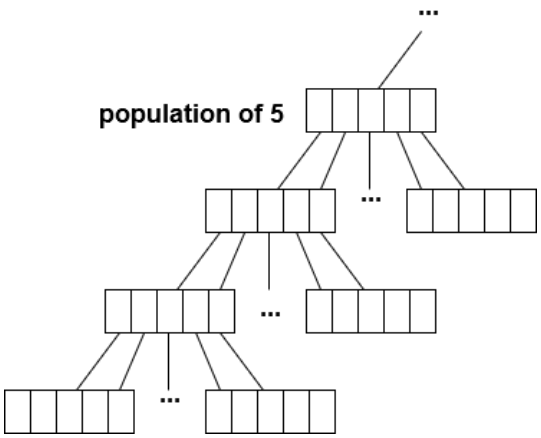


Figure 1. Dynamic chain hirarchy used in population management

extension of the VRP where we include drones in the fleet, therefore it is a heterogenous fleet VRP (HFVRP), but a special case because drones only have a capacity of only one parcel. In PDSVRP the drones start from the depot, deliver the package, and return to the depot for another package. Since a drone always must go back-and-forth to the depot the total distance will always be higher with a drone compared to a car. Therefore, we minimize the makespan in PDSVRP or the completion time from the first vehicle that leaves the depot to the last vehicle that finishes in the depot. Because drones only deliver one package, the order of the deliveries for drones does not matter, and each delivery has a fixed duration, we can transform the planning of the drones into an identical parallel machine scheduling problem (IPS). Now we have two sub-problems the IPS for drones and the VRP for trucks. The IPS has a great greedy solution method, longest-processing-time-first (LPT) (Graham, 1969), that gives close to optimal solutions in linear time. This way we can treat the problem as a relaxation of the VRP where not every location needs to be visited, this is known as the team orienteering problem (TOP). This way we can schedule in the left-over free locations for drones and efficiently get the routes with LPT. So, we can now optimize the makespan of the entire system at once in an efficient manner. A solution consists of a list of routes for trucks and a set of free locations that are scheduled in for drones.

3. Hybrid genetic search with drones (HGSD) for PDSVRP

To solve PDSVRP we use a Hybrid genetic search inspired by (Vidal et al., 2012) this is a genetic algorithm to solve (multi-depot/periodic) capacitated VRP (cVRP). The genetic algorithm uses a combination of mutations and crossovers in an evolutionary manner over a population of solutions. We use advanced population management and diversification management to guarantee a broad search over the solution-space. For the population management we use a chained hierarchy where leaves are random restarts that get combined into a new population. We have a main population with our best solutions where we keep adding solutions from a population of random restarts into each level of the hierarchy, forming a chain as illustrated in figure 1. Each level has a max number of iterations to start the new level. This dynamic population management is also made to easily allow parallelization as we can easily calculate the random restarts on a different thread. Diversification management is used to keep our population diverse; a diversification score is calculated between each pair of solutions in our population, based on the number of predecessors and successors each location has in common between the solutions. The distance is the percentage of different successors and predecessors. We get a diversity contribution of P, ΔP by taking the average of the n lowest diversity scores of all pairs of solutions including P. We then use a biased fitness score BF of a solution, we calculate it as follows for solution P:

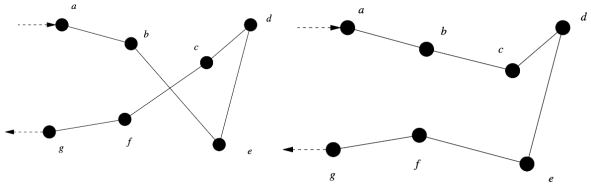


Figure 2. Illustration of 2-OPT

$$BF(P) = (1 - f) * fitness(P) + f * fitness(P) * \Delta P$$

The f is a constant percentage that gives weight to the importance of the diversity in our population.

We used numerous mutations with each an own heuristic to efficiently search over the useful parts of the solution-space. These mutations are efficient, effective, and scalable, so we can do a ton of mutations and see what gives an improvement. What follows is an overview of the mutations used:

- **Add:** Inserts at a random insertion position p of a route, n free locations that are closer than the average distance/location of all routes.
- **Remove:** Removes a sequence of n locations from a random route with a bad distance/location ratio and makes them free locations.
- **Swap-free:** Exchanges a random free location fl with a location l in a route, where l and fl are closer than the average distance/location of all routes.
- **Swap-pos:** Changes the position of a random sequence

of n nodes in a random route to the best positions in the route for that sequence.

- **2-OPT:** Heuristic by (Clarke & Wright, 1964) that removes crossings from a single route. (illustrated in figure 2)
- **cross-over mutation:** Exchanges a random sequence of a random route with a sequence from another route where the start and stop positions of the random sequences are close to each other.
- **Cross:** Extension of 2-OPT that finds different routes that cross and removes the crossing as an inter-route variant of 2-OPT.
- **Cross-over:** Exchanges a random sequence of a random route of the parent solution, into the route of the child solution on the start location of this sequence.

As a fitness function we use the biased fitness function that minimizes the makespan. The problem with only minimizing the makespan is that other advancements in routes different from the longest route are not reflected in the fitness score. Therefore, we made a fitness function that also reflects improvements in the deliveries/distance of the routes. Our fitness function becomes the following:

$$fitness(P) = \sum_{r \in routes} \frac{|r|}{Dist_r + f * makespan}$$

Here $routes$ is the set of all routes of the trucks and drones, $|r|$ is the number of locations in the route and $Dist_r$ the distance of the route, f is a weight to give more importance to the makespan. We plug this into the biased fitness function, and this gives the score we use to rank our population and choose the survivors. For the solution with the highest fitness, we ignore ΔP so it will always be on top of the population.

4. Online HGSD for solving the dynamic PDSVRP

When we want to use drones to facilitate same-day delivery, we will have to make an online variant of HGSD that can manage new incoming orders and remove locations when they are being handled. We use the term dynamic in the sense that the problem can change during calculations. When a route gets driven, it is fixed and will no longer change, therefore a route can be removed from calculations when it is started. A big added complexity is that now a vehicle can drive multiple routes. As a fitness function we no longer use the makespan, as the total time of all operations can no longer be calculated. Since a vehicle can do multiple routes, we can also wait to serve locations until the next route. Therefore, we optimize the deliveries/hour, with the objective to deliver as many parcels as possible in the given time. A problem however is that if we would just optimize deliveries/hour, small very efficient routes would almost always be chosen. Therefore, we add a starting penalty p_{start} so we look to the efficiency over at least p_{start} hours. Since we work with time, we also need truck and drone speed and load and deliver penalties (p_{load}, $p_{deliver}$).

$$time_r = dist_r/v_t + |r| * p_{deliver} + p_{load} + p_{start}$$

$$fitness_{truck} = \sum_{r \in routes} \frac{|r|}{time_r}$$

$$fitness(P) = fitness_{drone} + fitness_{truck}$$

$fitness_{drone}$ is calculated by adding the n drones with shortest delivery duration. The sum of the durations needs to be as long as possible but shorter than the current makespan, then the fitness of drones is given by $fitness_{drone} = \frac{n}{makespan}$. The other locations are for the next shift and not included in the current shift's fitness. Drones deliver the free locations from close to far. Because they come to the depot every time, the route can always change. They always take the closest free location of the current best solution. For a truck we always give the longest route that is not driven yet from the chosen solution.

When making the algorithm online we need to be able to adapt the problem dynamically. We can add locations by adding them in as a free location for all solutions in the current population. The problem this gives is that the fitness might change. Recalculating each fitness is too expensive. Adding a free location can only increase the deliveries/hour for a route, so we can class the location as not scheduled and the fitness does not change. For our best solution we however do recalculate our drone scheduling to get the exact best fitness. When we need to pick a route, we first choose the current best solution and freeze it, then we distribute the routes from this solution amongst the next trucks that need new routes. When we choose a solution, we start new calculations with all free locations from the chosen solution. When we remove a location because a drone needs a new customer, we take the closest free location of the current best solution. The fitness of a solution can however change if we remove a location. As we already mentioned recalculating to too expensive, but what we can do is make an over-approximation. The fitness of drones is calculated by $\frac{|r|}{2*makespan+p_{start}}$ where $|r|$ is the number of locations the drone does in the makespan. So, if we remove a location of a drone the fitness can go down with a maximum of $\frac{1}{2*makespan+p_{start}}$ and for a route it will also approximately go down with this value. So instead of recalculating, every fitness we can just remove the location from every solution and lower the fitness of every solution by $\frac{1}{2*makespan+p_{start}}$.

In a next mutation the fitness will then again be exact. we try to over approximate a little so, that way new mutations can have higher scores which also adds to the diversity.

Another problem is how do we stop? If we always create new routes for m trucks the locations always get split in m, if however, the end of the day is near, and some trucks are driving their last route we want trucks that start a new route to take as many parcels as they still can and not divide the parcels between m. When we start new calculations, we already know the approximated time the trucks will arrive for a new route. If this time is after the time limit, we know they will not need an extra route, so we can start the calculations with less routes. This way we can fully distribute the last locations amongst the last routes, that way we can make a clean stop.

5. PDSVRP test method & results

In literature there are multiple testing methods that are used in PDSVRP, there is not yet consensus about how we evaluate PDSVRP solutions. Firstly, there are multiple variants with different objective functions, the most common ones are the makespan as used by (Murray & Chu, 2015) another common fitness function is minimizing the logistics cost. But even in the case of minimizing the makespan there is no consensus on how we calculate this. To evaluate PDSVRP, firstly, we need to make a difference between trucks and drones, and secondly, we need to determine drone uneligible locations. In many papers the difference between drones and trucks is expressed by letting drones use Euclidean distance and trucks Manhattan distance. Furthermore, drones also have a different speed, sometimes by a factor and sometimes by different speed in km/h but this can also be reduced to a factor. Also currently choosing uneligible routes is done randomly or done by a method that chooses the first 30% based on index and the last 70% by farthest distance from depot. Randomly choosing uneligible locations makes the solutions not objectively reproducible, and choosing farthest locations adds a heuristic in the testing instance which is not ideal. The biggest problem with these testing methods in literature are that Manhattan distance has multiple routes with the same length between two points, therefore when we want to visit a far location, we can visit multiple locations between these locations without increasing the distance by only going in the direction of the far location (illustrated in figure 3). Therefore, a lot of the best makespans are just the Manhattan distance to the farthest uneligible location. This gives ugly solutions; therefore we propose a new objective testing method that uses following steps:

- Choose a cVRP instance (preferably from CVRPLIB)
- Choose drone uneligible locations: For K% drone eligible locations take the $\lfloor n * (K-1) \rfloor$ locations with the highest capacity and lowest index.
- Take the depot of the cVRP problem
- Drones and Trucks both use Euclidean distance.
- Minimize the distance of the longest route (makespan) with a distance-reducing speed factor for drones ($\frac{distance}{speed_{drone}}$).
- All locations are within drone range
- There is no capacity constraint on trucks

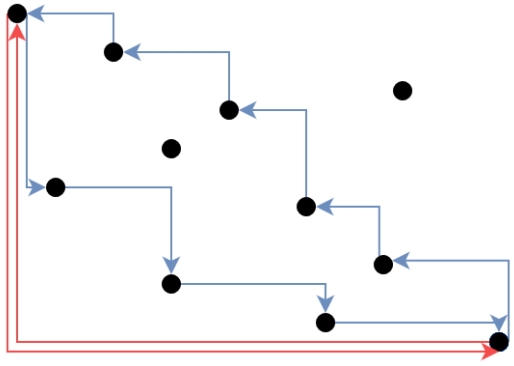


Figure 3. The big problem with Manhattan, blue route has the same Manhattan distance as the red route

Our calculations were done on the UGent HPC (2 x 18-core Intel Xeon Gold 6140 (Skylake @ 2.3 GHz), 88 Gb RAM (4Gb used)) with 1 hour of caluclation time for the instances in tables 1, 3, 4 and 8 hour for the instances in table 2. We tested our solutions against the current best solutions from (Raj et al., 2021) and (Saleu et al., 2022). We simulated the tests with the fitness function and constraints used in the referenced paper. Our results compared to Raj et al. are in table 4 and our results against Salue et al. are in 1. We were able to make improvements for a lot of instances. Especially for bigger instances our improvements are verry high. For instances with 250+ locations there were no solutions yet because most algorithms were not able to get decent results on these instances. We provide our solutions of larger instances 250-1000 nodes in table 3 an example is gives in figure 4 and of very large-scale instances 4.000-11.000 in table 2.

6. Conclusion

Drone delivery is a remarkably interesting and hot topic in research right now and could potentially become a huge revolution in the transport industry. With projects from major companies like Amazon with prime air (Amazon) and Google with Wing (Wing), drone delivery could be a reality sooner than you think. The research for algorithms

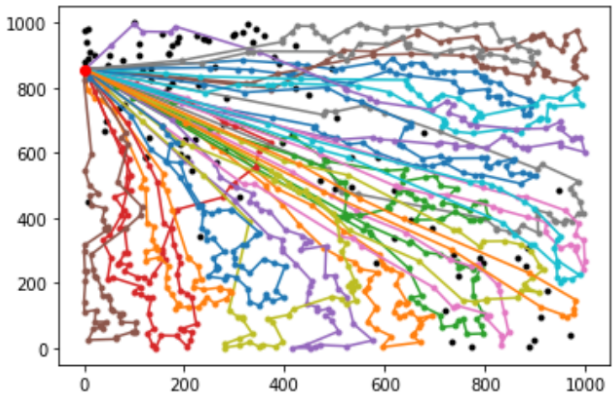


Figure 4. Solution for X-n1001-k43 80% drone eligible with black dots done by drones

to optimize this novel problem are still in an early stage with a lot of new papers in upcoming journals. Currently the solutions are still of varying quality and there are not yet standards or common practices among the researchers. In this paper we researched the problem and adapted one of the best performing cVRP algorithms hybrid genetic search form (Vidal et al., 2012) to be able to efficiently work with drones. The result beat the current state-of-the-art in PDSVRP solutions and even beat some solutions in PDSTSP. We also worked out a way to extend the algorithm to work online for the dynamic problem enabling the algorithm to also work in a same-day delivery environment. We also gave critique on the current way of testing PDSTSP and PDSVRP solutions and produced a new objective testing method for PDSTSP/PDSVRP. PDSVRP is new trendy part of VRP research where there is still a lot of opportunity to do research and potentially work on the next revolution within parcel delivery, to end in the words of Amazon "It would be easy to say, the sky is the limit, but that's not exactly true anymore, is it?".

Table 1. Results PDSmTSP compared against (Saleu et al., 2022)

Instance	n	#D	#V	BKS	HGSD	gap
CMT1	50	3	2	166	166	-0.0 %
CMT2	75	5	5	130.23	128.774	-0.95 %
CMT3	100	4	4	184	184	-0.0 %
CMT4	150	6	6	160.38	154	-4.14 %
CMT5	199	9	8	138	134	-2.99 %
E-n51-k5	50	3	2	168	166	-1.2 %
E-n76-k8	75	4	4	154	152	-1.32 %
E-n101-k8	100	4	4	184	184	-0.0 %
M-n151-k12	150	6	6	154	152	-1.32 %
M-n200-k16	199	8	8	144	138	-4.35 %
P-n51-k10	50	5	5	111.07	110	-0.97 %
P-n55-k7	54	4	3	126	126	-0.0 %
P-n60-k10	59	5	5	114	114	-0.0 %
P-n65-k10	64	5	5	126	124	-1.61 %
P-n70-k10	69	5	5	128	128.253	+0.2 %
P-n76-k5	75	3	2	200	200	-0.0 %
P-n101-k4	100	2	2	342	342	-0.0 %
X-n110-k13	109	7	6	1864	1812	-2.87 %
X-n115-k10	114	5	5	2258	2150	-5.02 %
X-n139-k10	138	5	5	2492	2376	-4.88 %

References

Amazon. Amazon prime air. https://www.amazon.com/Amazon-Prime-Air. Online; accessed 28 January 2022.

Clarke, G. and Wright, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

Dantzig, G. B. and Ramser, J. H. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

Graham, R. L. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

Mbiadou Saleu, R. G., Deroussi, L., Feillet, D., Grangeon, N., and Quilliot, A. An iterative two-step heuristic for the parallel drone scheduling traveling salesman problem. *Networks*, 72(4):459–474, 2018.

Murray, C. C. and Chu, A. G. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

Raj, R., Lee, D., Lee, S., Walteros, J., and Murray, C. A branch-and-price approach for the parallel drone scheduling vehicle routing problem. *Available at SSRN 3879710*, 2021.

Saleu, R. G. M., Deroussi, L., Feillet, D., Grangeon, N., and Quilliot, A. The parallel drone scheduling problem with multiple drones and vehicles. *European Journal of Operational Research*, 300(2):571–589, 2022.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

Wing, G. How it works. https://wing.com/how-it-works/. Online; accessed 29 November 2021.

Table 2. large-scale PDSVRPD

Instance	n	#D	#V	makespan
Leuven2	4000	35	35	3319.08
Antwerp2	7000	60	60	4466.41
Ghent2	11000	95	95	3386.71

Table 3. Results PDSVRP

Instance	#D	#V	HGSD 80%	HGSD 100%	gap %	HGSD Manh
X-n139-k10	5	5	1844.74	1795.57	-2.66 %	2164
X-n275-k28	14	14	1050.08	1040.16	-0.94 %	1528
X-n439-k37	19	18	1309.90	1290.09	-1.53 %	1856
X-n513-k21	11	10	1757.50	1710.78	-2.73 %	2208
X-n627-k43	22	21	2801.00	2563.95	-9.24 %	3960
X-n716-k35	18	17	1950.77	1786.24	-9.21 %	2718
X-n783-k48	24	24	2487.93	2274.74	-9.37 %	3592
X-n837-k142	71	71	2287.9*	1954.04	-17.08 %	3282
X-n916-k207	104	103	2680.47*	2026.65	-32.26 %	3920
X-n957-k87	44	43	1640.85	1569.79	-4.53 %	2436
X-n979-k58	29	29	2697.49	2600.55	-4.04%	3992
X-n1001-k43	22	21	2542.55	2405.77	-5.36 %	3574

Table 4. PDSTSP & PDSVRP results for TSPLIB instances from Salue et al. (Mbiadou Saleu et al., 2018)

		1			2			3		
Instance	#D	BKS	HGSD	gap %	BKS	HGSD	gap %	BKS	HGSD	gap %
att48	2	28686	**25695.5**	-10.42	17032.0	17196	0.96	14062	**13340**	-5.13
	4	28610	**20276**	-29.12	16500.0	**13811.6**	-16.29	13394	**12990**	-3.02
	6	28610	**20182**	-29.45	16500.0	**13246**	-19.72	13394	**12624**	-5.75
berlin52	2	5290.65	5330	0.74	3328.2	3348.78	0.62	2995.0	**2945**	-1.67
	4	5190.00	5190	0.0	2995.0	**2945**	-1.67	2625.0	2685	2.29
	6	5190.00	**5180**	-0.19	2995.0	**2945**	-1.67	2625.0	2675	1.90
eil101	2	456.00	473	3.73	305.4	**302**	-1.11	235.0	**229**	-2.55
	4	346.68	395	13.94	253.7	**252**	-0.67	200.1	204	1.95
	6	314.0	395	25.79	215.3	224	4.04	181.0	186	2.76
gr120	2	1188.51	1212	1.97	764.0	**754**	-1.31	597.0	**584**	-2.18
	4	946.04	1000	5.70	646.0	**644**	-0.31	528.0	**522**	-1.14
	6	851.4	927	8.88	581.2	**580**	-0.21	527.2	**476**	-9.71
pr152	2	70244	73630	4.82	48967.0	**41566.6**	-15.11	48135	**33590.1**	-30.22
	4	59772	64773.1	8.37	52353.0	**38672.6**	-26.13	43024	**32507.9**	-24.44
	6	56262	62202	10.56	50854.4	**35509**	-30.18	41324	**30807**	-25.45
gr229	2	1673.72	1742.09	4.08	1403.7	**1008.54**	-28.15	1145.4	**698.94**	-38.98
	4	1518.62	**1484.02**	-2.28	1346.2	**916.94**	-31.89	1132.3	**647.9**	-42.78
	6	1467.76	**1362.46**	-7.17	1327.6	**867.48**	-34.66	1130.0	**631.54**	-44.11

Vulgariserende samenvatting

In deze scriptie hebben we onderzoek gedaan naar het optimaliseren van het leverproces van pakjes. Hierbij hebben we bekeken hoe we drones kunnen inzetten bij het leveren van pakjes en welke winst dit zou kunnen geven. In 2013 kondigde Amazon aan dat ze bezig waren met onderzoek naar drone delivery, het leveren van pakketjes via drones. Er werd voorspeld dat dit potentieel een kostenbesparing van 80% in het lever proces zou kunnen betekenen en levertijden kan reduceren naar 30 minuten. Drone delivery kan dus een volgende revelatie zijn in de manier waarop we onze pakketjes krijgen. De opstelling van het probleem dat we proberen oplossen is als volgt: we hebben een vloot bestaande uit een aantal wagens en een aantal drones die allemaal aan een distributie centrum staan die pakketten verdeelt over allerlei locaties rondom dit centrum. Wat we proberen optimaliseren is de route die de wagens en drones zullen afleggen om de pakketten te leveren. Hierbij proberen we de totale tijd van de leveringen van het starten van het eerste voertuig tot het stoppen van het laatste te minimaliseren, dit wordt ook wel de makespan van het probleem genoemd. Hierbij kunnen drones slechts één pakje per keer dragen en moeten deze erna telkens een nieuw pakje halen uit de depot, en kunnen wagens meerdere pakjes per route meenemen. Dit probleem wordt in de onderzoekswereld het vehicle routing problem (VRP) genoemd, en wordt standaard enkel met een vloot van wagens opgelost. In het standaard geval wordt de som van de afstanden van de routes geminimaliseerd, maar bij drones is dit minder relevant en is de makespan de meest logische keuze. De variant waar we drones gebruiken die onafhankelijk van de wagens werken noemen we het parallel drone scheduling vehicle routing problem (PDSVRP) en werd in 2015 voor het eerst onderzocht door Murray en Chu [48].

Voor het oplossen van dit probleem hebben we een methode ontwikkeld die start van een aantal initiële routes en kleine slimme aanpassingen doet aan deze routes en kijkt of door deze aanpassing onze score verbeterd is. Deze score berekenen we aan de hand van de makespan, maar ook de algemene kwaliteit en efficiëntie van de route. We houden de aangepaste oplossingen met de beste scores bij in een populatie. We nemen vervolgens voortdurend een willekeurige route uit deze populatie, nemen een kopie om een aanpassing op te doen, en plaatsen het resultaat van de aanpassing terug in de populatie. Onze populatie heeft een maximumaantal oplossingen, als deze vol is verwijderen we de oplossingen met de laagste score. We doen dit met een percentage survivors, wat het aantal oplossingen is met de hoogste score die door mogen naar de volgende generatie. Zo blijven telkens de beste oplossingen over en werken we hierop verder. Dit is op een gelijkaardige manier zoals evolutie en mutaties in de natuur ook werken, daarom wordt dit een genetisch algoritme genoemd. De aanpassingen die we doen op routes kunnen acties zijn zoals twee locaties op verschillende routes omdraaien of een locatie weghalen uit een route of een locatie toevoegen aan een route. We hebben zo 8 acties gedefinieerd waarmee we aanpassingen kunnen maken aan de routes. We moeten er enkel voor zorgen dat onze populatie goede diverse oplossingen blijft bevatten en zo kunnen we door aanpassingen te blijven doen goede oplossingen bekomen na verloop van tijd. Dit geeft ons dan een benaderde goede oplossing, die hopelijk dicht zal liggen bij de optimale oplossing. We kunnen deze oplossingen blijven aanpassen en blijven verder zoeken, meestal blijven we zoeken voor een gegeven tijd of zolang we verbeteringen vinden.

We breiden het probleem ook uit waarbij locaties tijdens de berekeningen toegevoegd of verwijderd kunnen worden en routes ook gereden kunnen worden tijdens de berekeningen. Er kunnen meerdere routes per koerier gereden worden en het probleem verandert continu. Op deze manier kunnen we same-day delivery mogelijk maken met dit model, waarbij we op dezelfde dag van bestelling nog het pakket kunnen leveren. We kunnen echter de totale tijd niet meer weten tijdens de berekeningen aangezien er nog locaties kunnen bijkomen daarom maximaliseren we hier niet de makespan, maar wel het aantal leveringen/uur.
Wanneer we onze resultaten vergelijken met mensen die hetzelfde probleem onderzocht hebben zien we dat in veel gevallen we een verbetering hebben kunnen maken en nieuwe ondergrenzen voor de makespan vinden van enkele problemen. Vooral op grote problemen haalt ons algoritme veel betere

resultaten. Onze scriptie is ook de eerst die oplossingen uitbrengt voor grote problemen met 250-11.000 locaties. We hebben ook kritiek geuit tegenover de huidige manier van testen en evalueren van de oplossingen, en een nieuwe methode ontwikkeld voor het objectief evalueren van oplossingen van het parallel drones scheduling VRP.
We hebben dus een methode uitgedacht die competitief is met de beste oplossingsmethode van dit probleem, en onderzoek gedaan naar de volgende stappen van drone delivery. We hebben hierbij ook de manier waarop we dit kunnen programmeren gedetailleerd beschreven en ook onze eigen code openbaar gemaakt zodat andere onderzoekers hierop kunnen verder werken en de resultaten verifiëren.

Dankwoord

Deze masterproef is de conclusie van mijn 5 jaar lange studies aan de Universiteit Gent, en is de kers op de taart van mijn opleiding Master of Science in de informatica. Al sinds jongs af aan ben ik geboeid geweest door computers, programmeren, problem solving en optimalisatie. Ik ben dan ook zeer dankbaar dat ik mij een volledig jaar heb kunnen storten op een boeiend optimalisatie probleem als het vehicle routing probleem. Ik wil dan ook allereerst graag mijn promotor Veerle Fack bedanken om mij te begeleiden in dit onderzoek en de vrijheid te geven er mijn eigen ding mee te doen. Hierdoor heb ik er een praktische, futuristische toepassing van kunnen maken en heeft het onderzoek mijzelf ook zeer hard weten te boeien. Ik wil haar en Gunnar Brinkmann ook graag bedanken om hun fascinatie voor algoritme en datastructuren door te geven tijdens mijn opleiding. Ik wil ook Universiteit Gent, de opleiding informatica en mijn professoren en assistenten bedanken voor het kwalitatief onderwijs dat ik heb mogen genieten in de afgelopen jaren, en mij de mogelijkheid te geven voor het gebruiken van de HPC voor dit onderzoek.

Verder wil ik graag mijn familie en vrienden bedanken voor hun onvoorwaardelijke steun doorheen het jaar. Ik wil ook graag mijn vrienden in de opleiding bedanken om de afgelopen 5 jaar een pak aangenamer te maken en mij te helpen doorheen deze tijd. Ik wil nog in het bijzonder Steven Van Overberghe en Niels Dossche bedanken om mijn passie voor problem solving en informatica aan te wakkeren, en mij altijd paraat te staan met hulp wanneer ik dit nodig had. Ik wil ze ook bedanken voor het leveren van nuttige feedback op deze scriptie.

Glenn Feys, juni 2022

Toelating tot bruikleen

De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef.

Glenn Feys, juni 2022

Inhoudsopgave

Lijst van afkortingen

Afkorting	Betekenis
ADC	Adaptive diversity control
BKS	Best known solution
CO	Cross-over
cVRP	Capacitated vehicle routing problem
DVRP	Dynamic vehcile routing problem
DVRPD	Dynamic vehicle routing problem with drones
FSMVRP	Fleet size max vehicle routing problem
FSTSP	Flying sidekick traveling salesman problem
GA	Genetic algorithm
HFVRP	Heterogenous fleet VRP
HGS	Hybrid genetic search
HGSADC	Hybrid genetic search with adaptive diversity control
HGSD	Hybrid genetic search with drones
HVRPD	Heterogeneous VRP with vehicle dependent routing costs
HVRPFD	Heterogeneous VRP with fixed costs and vehicle dependent routing costs
IPS	Identical parallel machine scheduling problem
LPT	Longest-processing-time-first scheduling
MDPVRP	Multi-depot periodic vehcile routing problem
MDVRP	Multi-depot vehicle routing problem
mFSTSP	Multiple flying side-kicks traveling salesman problem
MFVRP	Mixed fleet vehicle routing problem
MILP	Mixed-integer linear programming
mTSP	Multiple traveling salesman problem
PDSmTSP	Parallel drone scheduling multiple traveling salesman problem
PDSTSP	Parallel drone scheduling traveling salesman problem
PDSVRP	Parallel drone scheduling vehicle routing problem
PDSVRPTW	Parallel drone scheduling with time windows
PFA	Parametric policy function approximation
PTP	Profitable tour problem
PVRP	Periodic vehicle routing problem
SDD	Same-day delivery
SDDPHF	Same-day delivery routing problem with heterogenous fleets
SISSRs	Slack induction by string and sweep removals
STSP	Selective traveling salesman problems
TOP	Team orienteering problem
TSP	Traveling salesman problem
TSP-D	Traveling saleman with drones
UAV	Unmanned Aerial vehicles
VRP	Vehicle routing problem
VRPD	Vehcicle routing problem with drones
VRPMT	Vehicle routing problem with multiple trips
VRPTW	Vehicle routing problem with time windows

Abstract

De transportsector is al jarenlang een van de grootste en meest vervuilende sectoren ter wereld. Met de globalisering en de groei van e-commerce zien we een grote toename in het belang van deze sector in het transporteren van goederen. De laatste jaren is er ook veel onderzoek gedaan om enerzijds het transport van goederen en anderzijds de routes voor het leveren te optimaliseren. Hierbij wordt het gebruik van drones vaak gezien als mogelijke revolutie in de last-mile delivery, de laatste kilometers die een pakje moet afleggen tussen het verdeelcentrum en de bestemming. Ondanks de vele research naar de technische en praktische kant van drone delivery, is de research naar algoritmes voor drone delivery en het incorporeren van drones in vehicle routing problems (VRP) pas echt in gang geschoten na de paper van Murray en Chu [48] in 2015 over "the flying sidekick traveling salesman problem". Veel van de huidige studies naar VRPs met drones spitsten zich vooral toe op het optimaliseren van relatief kleine problemen die in benchmark datasets te vinden zijn, dit gaat over problemen van rond de 50-200 locaties. Substantiële research naar large-scale vehicle routing problems met drones is er voorlopig nog niet, maar deze is zeer relevant aangezien distributiebedrijven met problemen van deze grootte werken. In deze scriptie proberen we dit gat te dichten en bespreken we large-scale vehicle routing problems met parallel ingeplande drones ofwel parallel drone scheduling vehicle routing problem (PDSVRP). Waarbij we de de totale duur van de operaties, de makespan, proberen te minimaliseren. We bekijken de huidige oplossingsmethoden en stellen een competitieve hybrid genetic search met drones (HGSD) oplossingsmethode voor die geïnspireerd is op HGS voorgesteld door Vidal et al. [66]. Hiermee behalen we oplossingen die efficiënter zijn voor large-scale PDSVRPs en is er een verbetering op de huidige state-of-the art en behaalden we nieuwe best known solutions. Daarnaast bespreken we hoe drones een oplossing kunnen bieden om same-day delivery mogelijk te maken. We bespreken de extra aspecten die komen kijken bij same-day delivery en breiden onze oplossingsmethode verder uit om ook op een online manier het dynamische VRP met drones op te lossen om same-day delivery te faciliteren. Hierbij kan tijdens de berekeningen nieuwe lever-locaties toegevoegd worden, en routes gestart worden. Tot slot bespreken we een nieuwe objectieve testmethode voor PDSVRP en vergelijken we oplossingen van de huidige state-of-the-art oplossingsmethoden met onze genetische methode en kijken we welke kost we kunnen besparen door drones in te zetten in het lever proces.

Hoofdstuk 1

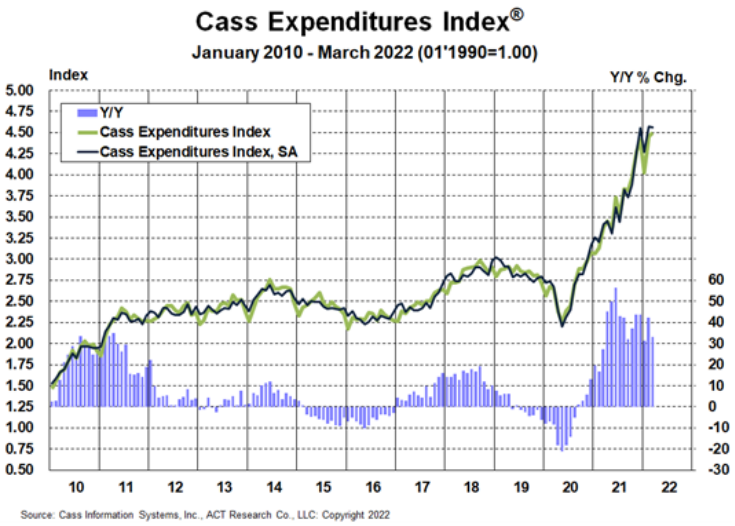
Introductie

Figuur 1.1: Amazon prime air

Onbemande luchtvaartuigen, beter bekend als drones, hebben de laatste tijd meer en meer aandacht gekregen in de transportsector, meer specifiek voor last-mile delivery. Dit komt niet als een verrassing, aangezien de laatste kilometers die een pakket moet afleggen ook de duurste zijn met grote marge. Last-mile delivery is de laatste stap in het lever proces waarbij het pakket van het distributiecentrum naar de klant vervoerd wordt, vaak gaat meer dan 50% van de lever kosten ook naar deze stap [31]. Hierdoor zijn grote bedrijven zoals Google[70], Amazon[3] en DHL[29] continu bezig met onderzoek naar het optimaliseren van deze laatste kilometers, waarbij zij ook hun eerste stappen gezet hebben richting drone delivery. Dit vooral om de kost van de lever operaties te minimaliseren, maar ook om levertijden te kunnen inkorten. Wanneer we levertijden namelijk verder kunnen verbeteren kunnen we mogelijks overgaan naar een systeem van same-day delivery. Hierbij kunnen klanten nog de dag van bestelling hun bestelde goederen ontvangen. Amazon beweert zelfs een stap verder te kunnen gaan en devilvery binnen 30 min te voorzien met prime air [3]. De laatste jaren zijn drones, en andere onbemande voertuigen zoals zelfrijdende robot wagentjes [35] naar voor geschoven voor de toekomst van last-mile delivery. Dit omdat het grootste knelpunt in de transportsector een tekort aan goedkope manschappen is. De laatste jaren neemt het aantal verzonden pakketten ook jaar na jaar toe. Uit het global parcel delivery market insight report 2021 [41] blijkt dat de globale online retail sales jaar na jaar met ongeveer 25% stijgen. Vele distributiecentra en koeriersbedrijven kunnen deze snelle stijging

niet volgen waardoor ze vaak bij drukke periodes grote achterstanden oplopen. Deze achterstanden zijn voornamelijk het gevolg van een tekort aan mankracht, de shift naar onbemande voertuigen kan hier een mogelijke oplossing voor bieden.

Wetgeving is echter nog een groot obstakel voor drone delivery Murray et al. [48] wist het volgende te vertellen: Amazon geeft aan dat 86% van de leveringen die ze uitsturen minder weegt dan de 2,5 kg maximale draag capaciteit van de drone. Ze melden ook dat de grootste hindernissen er nog zijn op het gebied van technologie, maar zeker ook wetgeving. Deze moeten eerst overwonnen worden vooraleer drone delivery op grote schaal uitgerold kan worden. In de Verenigde staten verbieden FAA-regels momenteel het inzetten van drones voor commerciële doeleinden. Verder bestaan er ook regels over de mobiliteit van drones, zo mogen drones volgens de FAA niet hoger gaan dan 120 meter en mogen ze niet komen binnen de "line-of-sight"van piloten. Momenteel staan er ook restricties op de range van een drone en dient er ook altijd een persoon een drone te bedienen. Er bestaan echter al uitzondering op deze regels, zo heeft gas en oliebedrijf BP reeds groen licht gekregen voor het vliegen van drones overzee in Alaska. Dit geeft bedrijven zoals Amazon goede hoop dat de FAA zijn drone policy aan het versoepelen is en mits wat lobbying er mogelijkheid bestaat voor ook groen licht voor bedrijven als Amazon voor het testen en uitrollen van drone delivery in de US. Andere landen zoals Australië en Ierland hebben een soepelere drone wetgeving en daarom worden veel testen momenteel in landen met een mildere drone wetgeving uitgevoerd, en zouden deze ook als eerste drone delivery op grotere schaal kunnen zien.



Figuur 1.2: Cass freight index maart 2022 expenditures, is een index van de gemiddelde bezorgkosten van een vracht in de VS.

Bij een bevraging over de keuze voor leveringsmethodes, door Joerss et al. [31] is gebleken dat wanneer de consument de keuze krijgt, de meeste consumenten de goedkoopste leveroptie verkiezen (70%), gevolgd door same-day delivery (23%). Als we kijken naar de huidige trends op de markt zien we dit ook duidelijk gereflecteerd worden, met gratis next-day delivery die vele webshops aanbieden. Uit de bevraging is gebleken dat same-day delivery dus zeker de moeite waard is wanneer dit goedkoop aangeboden kan worden. Maar goedkoop en snel gaan niet altijd hand in hand, we zien namelijk een grote stijgende trend in de leverkosten in figuur 1.2. Het is dus goed mogelijk dat gratis levering aanbieden niet meer mogelijk zal blijven voor bedrijven. Een snellere levering betekent over het algemeen ook meer manschappen nodig en een extra overhead voor de planning omdat er minder pakjes zijn om efficiënt in te plannen en de routes dus korter en minder efficiënt gemaakt kunnen

worden. Om same-day delivery uit te voeren met de huidige vloot van voertuigen zou het lever proces dus nog duurder kunnen worden wat dus goedkope levering onmogelijk zal maken en het ook moeilijk zal worden dit door te rekenen in de product kosten. Uit dezelfde ondervraging blijkt ook dat vooral voedsel en medicijnen niet online gekocht worden door te lange levertijden. Hier zou same-day delivery dus wel het gat kunnen dichten waardoor ook deze zaken online besteld kunnen worden.
In deze paper stellen we last-mile delivery door een gemengde vloot van wagens en drones voor als oplossing voor het mogelijk maken van same-day delivery op een kostenefficiënte manier. Hierbij gaat het vooral over het incorporeren van drones in huidige vehicle routing problems (VRP). We lossen het VRP dus op met een heterogene vloot van wagens en drones die parallel en autonoom van elkaar werken. We zoeken naar een efficiënte toewijzing van routes en locaties aan de vloot van wagens en drones. Hierbij proberen we de totale duur van de operaties, ook wel de makespan genoemd, te minimaliseren voor een gegeven vaste vloot grootte. Bij SDD bekijken we hoe we dit algoritme moeten aanpassen om het online te laten werken, zodat we hiermee ook het dynamische probleem kunnen oplossen waarbij op elk moment routes gereden kunnen worden en pakjes erbij kunnen komen. We bespreken daarnaast ook andere methodes van het incorporeren van drones in het VRP, en bespreken de verschillen in kost, efficiëntie en praktische aspecten.

In deze scriptie schetsen we eerst de problemen die we zullen oplossen door de verschillende delen ervan te bespreken aan de hand van een literatuurstudie, startend bij het standaard vehicle routing problem (VRP) en capacitated VRP (cVRP), gevolgd door het algemene heterogenous fleet VRP (HFVRP) waarbij verschillende voertuigen met verschillende capaciteit en eigenschappen mogelijk zijn in een vloot. Hierna bespreken we de speciale variant hiervan met drones, vehicle routing problem with drones (VRPD), dit probleem kent meerdere varianten die op een andere manier drones gebruiken bij het leveren. We vergelijken deze varianten op basis van hun research, efficiëntie en toepasbaarheid in de praktijk. We lichten ook de twee belangrijkste varianten van TSPs met drones toe: flying sidekick TSP (FSTSP), parallel drone scheduling TSP (PDSTSP) en hun VRP-uitbreidingen. We staven hierna onze keuze voor PDSVRP. Daarna kijken we naar het dynamic VRP (DVRP) en same-day delivery VRP (SDDVRP) waar het probleem constant aangepast kan worden door nieuwe bestellingen en gereden routes. Om uiteindelijk te komen tot dynamic VRP drone (DVRPD), waar we het dynamische VRP trachten op te lossen met behulp van drones. Hierna doen we een uitgebreide literatuur review over PDSTSP en PDSVRP de variant van het vehicle routing probleem dat we proberen op te lossen, en de online/dynamische variant hiervan. Hierna wordt onze genetische oplossingsstrategie besproken en toegepast op PDSVRP en hierna bespreken we hoe we deze kunnen uitbreiden zodat deze ook compatibel is om DVRPD op te lossen. Uiteindelijk bekijken we de resultaten en vergelijken we onze oplossingsmethode met de huidige state-of-the-art. We stellen ook een eigen experiment-/evalueer methode voor waarmee we op een objectieve manier PDSVRP oplossing met elkaar kunnen vergelijken.

1.1 Voordelen en nadelen van drones in VRP

Het toevoegen van drones aan een VRP kan vele voor- en nadelen hebben. In deze sectie maken we een analyse van de verschillende pros en contras bij het inzetten van drones voor het leveren van pakjes. We beginnen bij de voordelen: drones kunnen grotendeels autonoom werken, zo is er de grote kost van de werknemer die weg valt. Ze kunnen ook zeker in drukke gebieden zich sneller transporteren dan wagens, dit omdat ze in vogel vlucht kunnen vliegen, en geen last hebben van verkeer. Doordat ze geen deel uitmaken van het verkeer kunnen ze ook de drukte van het stedelijk verkeer verminderen, en de emissie van de transportsector in de steden verminderen. Drones kunnen ook 24/7 operationeel gehouden worden in vergelijking met de beperkte werkuren van koeriers. Een ander groot voordeel van drone delivery is dat slecht bereikbare plaatsen zoals eilanden ook snel behandeld kunnen worden door drones, zo deed DHL al verscheidene tests met drone delivery op het eiland Juist [29] in Duitsland vanop het vaste land. Maar het grootste voordelen blijven toch de lage kost voor het operationeel te houden van drones, en de lage ecologische voetafdruk.

Als we dan kijken naar de nadelen is een groot restrictie aan drones dat ze zeer afhankelijk zijn van het weer, in slechte weer omstandigheden zoals regen en hevige wind kan een drone niet operationeel gehouden worden, hierdoor is een toekomst met enkel drone delivery ook niet direct mogelijk. Drones hebben ook een beperkte range, ook al is dit tegenwoordig geen groot probleem meer met prototypes van Google en Amazon die ruimschoots 15-20 km behalen [70][3], wat genoeg is om een middelgrote stad als Antwerpen of Gent te faciliteren. Ook laadvermogen is een probleem bij drones, deze kunnen in huidige prototypes slechts één enkel pakket per keer vervoeren, en dit pakket heeft een beperkt maximumgewicht. Ook het lossen kan zeker op plaatsen als appartementsgebouwen een obstakel vormen. Hoe huidige test-projecten zoals Google Wing [70] hiermee omspringen is via een los systeem dat met een touw het pakketje laat neerdalen, op deze manier hoeft de drone niet te landen. Voor het probleem met appartementen heeft Amazon lever stations met QR-codes waar het pakket gedropt kan worden, zo kan bijvoorbeeld ook op het dak van een appartementsgebouw of in een tuin een lever locatie zijn. Er waren ook plannen voor het maken van lever boxen die via een signaal een drones kunnen detecteren, de box openen en na het leveren de box terug sluiten, zo kan zonder problemen 24/7 leveringen mogelijk zijn. De batterij van drones dient ook opgeladen te worden, dit is een extra tijds kost die elke toer meegerekend dient te worden. De mogelijkheid bestaat echter om de batterij hot-swappable te maken, waardoor de laadtijd drastisch vermindert kan worden ten koste van extra batterijen. Dit is ook het model waarmee wij zullen werken, wij rekenen dus geen extra tijd of kost aan voor het opladen van drones in de depot. Door hun beperkte range is een groot gebied voorzien van drone delivery ook een obstakel. Door de range kunnen drones maximaal 10km ver gaan wat betekend dat minimaal elke 20km er een lever station moet zijn om leveringen door drones in dit volledige gebied mogelijk te maken.
We zien dus dat de keuze voor drones dus ook gepaard gaat met enkele nadelen en er dus een analyse gemaakt dient te worden bij het inzetten van drones.

Hoofdstuk 2

Het vehicle routing probleem

Het vehicle routing probleem is een van de meest onderzochte combinatorische optimalisatie problemen van de afgelopen jaren. Danzig et al [14] besprak voor het eerst in 1959 het probleem in "The truck dispatching problem". Sindsdien zijn er reeds tal van papers geschreven over het probleem, en zijn tal van varianten van het probleem ontstaan, vaak met eigen nieuwe constraints en use-cases. Het vehicle routing problem is een generalisatie van het traveling salesman problem (TSP). Bij het TSP is het de bedoeling dat we vanuit een bepaalde start node een pad construeren dat door alle nodes gaat en eindigt bij de node waar we gestart zijn. Het is de bedoeling om dit pad zo efficiënt mogelijk te maken. Wiskundig uitgedrukt construeren we een zo efficiënt mogelijke hamiltoncykel voor de nodes. Het probleem kan gezien worden als een reiziger die een aantal locaties probeert te bezoeken met zo weinig mogelijk afstand af te leggen. Bij het vehicle routing problem laten we meerdere cykels starten vanuit de start node, en maken we dus routes voor meerdere reizigers. We proberen de locaties te verdelen onder de verschillende routes om zo alle locaties te bezoeken. Hierbij proberen we de totale kost voorgesteld door de som van de afstand van de routes te minimaliseren. Aangezien het TSP NP-hard is en het VRP een generalisatie is van het TSP, kunnen we ook besluiten dat het VRP ook een NP-hard probleem is en dus niet in polynomiale tijd op te lossen is.

Voor de verschillende varianten van het VRP zijn ook tal van oplossingsmethodes ontworpen. Er zijn exacte oplossingsmethodes ontworpen die bijvoorbeeld met behulp van branch-and-cut [42] en set partitioning formulation [1] grootte ordes van een honderdtal nodes kunnen behandelen [5]. Maar de meest populaire en onderzochte methode is toch bij uitstek de benaderende methodes die met behulp van heuristieken en meta-heuristieken tot benaderde oplossingen kunnen komen. Zeker voor varianten van het VRP zijn vaak bijna uitsluitend heuristische methodes die gebruikt worden voor het oplossen van het probleem. Met benaderende methodes kunnen we problemen van enkele honderden of zelfs duizenden nodes benaderd oplossen waarvan oplossingen vaak slechts op een fractie van een procent liggen van de optimaal bewezen oplossing. In deze scriptie zullen we enkele van deze meta-heuristieken bespreken en vergelijken met de voorgestelde genetische oplossingsmethode. Omdat veel praktische problemen grootte ordes hebben van enkele honderden locaties en de rekentijd beperkt is, worden meta-heuristieken vaak naar voor geschoven bij het oplossen van VRPs. Een marge van 1 tot 2 percent is ook niet zeer significant in praktische scenario's aangezien door externe factoren zoals verkeer en bereikbaarheid het verschil in theorie en praktijk vaak groter is dan deze 1-2 percent en de bereken tijd vaak beperkt is tot enkele minuten/uren.

2.1 Standaard vehicle routing probleem

Bij het klassieke vehicle routing probleem hebben we een set van n locaties $L = \{l_1, l_2, ..., l_n\}$. Deze worden gedefinieerd door een x en een y coördinaat $l_i = (x_i, y_i)$. Hiernaast hebben we ook één depot

l_0 die gespecificeerd wordt door een coördinaat. Een depot heeft een vooraf gedefinieerde vloot van m wagens $W = \{w_1, w_2, ..., w_m\}$. Wagens hebben een maximale capaciteit c_w die de maximale capaciteit pakjes voorstelt dat een wagen kan leveren op een route, hierbij is de capaciteit van elke wagen hetzelfde. Het is belangrijk te vermelden dat bij het klassieke VRP elke locatie slechts voor één meetelt in de capaciteit, als we toelaten dat locaties een hogere capaciteit hebben krijgen we het capacitated VRP (cVRP) waarover meer in sectie 2.1.1. In ons geval is dus het maximumaantal locaties in een route gelijk aan de capaciteit van een wagen c_w. Een oplossing voor het probleem bestaat uit een set van m routes $R = \{r_1, r_2, ..., r_m\}$ even groot als het aantal wagens in onze vloot. Hierbij bestaat elke route uit een geordende lijst van n locaties die we één voor één bezoeken op de route.

$$r_i = [l_1, l_2, ..., l_n]$$

Hierbij geldt dat:

$$0 \leq n \leq c_w$$

Voor de routes geldt verder nog dat elke locatie door precies één route bezocht wordt. Het doel dat we voor ogen hebben is het optimaliseren van deze routes. Hetgeen waarnaar we willen optimaliseren hangt af van het doel van het VRP. In het standaard VRP wordt de totale afgelegde afstand $dist_{tot}$ geminimaliseerd, wat de som is van de afstanden van de afgelegde routes.

$$dist_r = dist(l_0, r[0]) + dist(r[n], l_0) + \sum_{i=0}^{n-1} dist(r[i], r[i+1])$$

$$dist_{tot} = \sum_{r \in R} dist_r$$

Hierbij stelt de dist functie de afstand tussen twee locaties voor, in ons geval is dit vaak de Euclidische afstand maar dit kan ook een andere afstandmetriek of zelfs het wegennet zijn. Andere dingen waarnaar we kunnen optimaliseren is bijvoorbeeld de totale tijd wanneer elke route parallel gereden wordt, dit wordt ook wel de makespan van de routes genoemd. Hiervoor dient wel een notie van snelheid van een voertuig gedefinieerd te worden. Aangezien er geen tijd aangerekend wordt voor het behandelen van een locatie komt dit neer op de afstand van de langste route minimaliseren, hierdoor minimaliseren we de totale duur van de operaties van de routes, dit wordt ook de makespan van R genoemd. Maar wanneer we een heterogene vloot krijgen met verschillende snelheden van voertuigen of een verschillende reis kost is deze formule niet meer van toepassing. Deze formule gaat er ook van uit dat het aantal routes gelijk is aan het aantal koeriers.

$$dist_r = dist(l_0, r[0]) + dist(r[n], l_0) + \sum_{i=0}^{n-1} dist(r[i], r[i+1])$$

$$makespan(R) = \max(dist_r : r \in R)$$

Als we willen dat er een oplossing bestaat voor het VRP moet het aantal locaties $|L|$ kleiner zijn dan de maximumcapaciteit van de vloot.

$$|L| \leq m \cdot c_w$$

Anders zullen er meerdere routes gereden moeten worden door een koerier of blijven er locaties onbezocht. Het rijden van meerdere routes kan makkelijk toegevoegd worden aan het probleem door de oplossing uit te breiden met een extra route, deze stelt een 2e route van een bepaald voertuig voor. Dit probleem wordt het vehicle routing problem with multiple trips (VRPMT) genoemd. Bij VRPMT is de assumptie dat het aantal routes gelijk is aan het aantal voertuigen in de vloot wel verbroken. Dit heeft gevolgen voor onder andere bepaalde fitness functies die bijvoorbeeld de makespan minimaliseert.

2.1.1 Capacitated vehicle routing problem (cVRP)

Bij het vehicle routing problem zoals deze het meeste beschreven staat in de literatuur wordt bij elke locatie een extra parameter bijgehouden voor de vraag of capaciteit van de levering q_i en de capaciteit van een koerier c_w. Dit is ook het geval bij de paper van Danzig et al. [14]. Hier ging het over het bijvullen van tankstations doorheen het land. Hier was deze capaciteit q_i het aantal liters dat een tankstation nodig heeft, en c_w de maximumcapaciteit van de tanker. Hierbij gaan we ervan uit dat q_i een strikt positief natuurlijk getal is. We hebben ook nog steeds de constraint dat elke locatie precies één keer bezocht wordt, en dus moet een wagen altijd de volledige capaciteit van een locatie vervullen als hij deze bezoekt. Hierdoor kunnen we niet meer uitgaan van de assumptie dat het aantal bezochte locaties gelijk moet zijn aan de capaciteit van de wagen. Nu krijgen we dat:

$$|r| \leq c_w$$

en

$$c_w \geq \sum_{l_i \in r} q_i$$

Waarbij c_w de capaciteit voorstelt van een wagen en q_i de vraag van de levering op locatie i. Wanneer we dit toevoegen krijgen we een extra dimensie aan het probleem, dit creëert namelijk een extra bin-packing probleem om de wagens zo goed mogelijk te vullen met de capaciteiten van de leveringen. Hierdoor zijn ook nieuwe doelfuncties mogelijk die belangrijk zijn voor de efficiëntie van het VRP. Zo moet nu bijvoorbeeld bij sommige doelfuncties gekozen worden of we optimaliseren naar aantal locaties behandeld of capaciteit geleverd. Een voorbeeld hiervan is de doelfunctie die we later ook zullen gebruiken leveringen/uur, bij het standaard VRP is deze gelijk aan capaciteit/uur maar in cVRP is dit niet langer het geval en zouden we in sommige gevallen liever capaciteit/uur willen maximaliseren. De standaard doelfunctie waarnaar echter geoptimaliseerd wordt in het standaard cVRP probleem is wederom de totale afstand van de routes $dist_{tot}$ waarbij elke locatie bezocht wordt en geen enkele wagen over capaciteit gaat op een route. De formule van $dist_{tot}$ blijft dezelfde, we moeten enkel met de nieuwe contraint $c_w \geq \sum_{i=1}^{m} q_i$ rekening houden.

Bij cVRP wordt meestal gewerkt met een vloot van wagens met een vast aantal. In sommige varianten wordt er enkel een minimumgrens van aantal wagens gezet, maar kan men meer wagens gebruiken. Vaak is het efficiëntste om routes zo lang mogelijk te maken en dus zo weinig mogelijk wagens te gebruiken. Daarom worden in benchmark datasets van cVRP vaak met minimumaantal wagens gewerkt en staat de minimumaantal wagens gelijk aan het minimum aantal wagens die de capaciteit kan rondbrengen wat ook bijna altijd het aantal wagens van de beste oplossing zal zijn, hierdoor kan men het probleem ook oplossen voor vaste vloot.

Het VRP waarbij de capaciteit gedefinieerd wordt door het maximaal aantal locaties die bezocht kan worden, hiervoor beschreven als het standaard VRP, is een vereenvoudiging van cVRP waarbij elke levering een capaciteit van één heeft. Op deze manier valt het bin-packing probleem weg. Dit is een abstractie die kan gemaakt worden als lever groottes (bijna) niet verschillen, of er niet veel toe doen.

2.2 Vehicle routing problem met drones (VRPD)

We zien dat research naar oplossingen voor allerhande problemen met drones enorm gestegen is de afgelopen jaren. Zo zien we dat drones tegenwoordig reeds applicaties kennen in de filmindustrie, bij hulpdiensten, bij het onderhoud van hoogspanningskabels en tal van andere zaken. Voorlopig blijven de use-cases vaak beperkt tot bewakings- en filmtaken, maar de mogelijkheden met drones gaan zeer ver. Een van de grootste pluspunten van drones is voornamelijk hoe ze snel luchttransport kunnen aanbieden op een kostenefficiënte manier. Bij drone delivery wordt deze eigenschap ook ten volle benut. Bij vehicle routing problems met drones wordt gefocust op het combineren van een conventionele vloot van koerierwagens met drones, en de integratie van drones in dit probleem. Er zijn echter verschillende manieren waarop we drones kunnen incorporeren in het VRP.

In de literatuur zien we verschillende voorstellen voor het toevoegen van drones aan het VRP. Allereerst zien we het standaard geval waar drones vertrekken uit de centrale depot, hun pakket leveren en terug keren naar de depot [62] [48]. Hiernaast zien we ook nog een andere optie waar de drones als side-kick van een wagen werken [48] [69] [59]. De drone en truck gaan samen op route waarbij de drone stationair gevoerd wordt door de wagen en meerdere malen op de route kan opstijgen van de truck, zelfstandig een pakket leveren los van de truck, en erna terug landen op de truck. Een laatste optie die beschreven wordt in de literatuur is het opladen van een aantal drones in een truck, deze allemaal naar een locatie transporteren en daar vervolgens loslaten. Hierna landen de drones terug bij de truck en worden ze terug naar de depot vervoerd, dit wordt ook wel het mothership model genoemd en werd besproken door Poikonen et al. [52]. Veel van deze ideeën hebben zeker potentieel en kunnen voor betere resultaten zorgen bij het VRP in theorie, maar de praktische verwezenlijking op een efficiënte manier lijkt soms wat lastiger. In deze scriptie bespreken en vergelijken we de verschillende manieren uitvoerig.

Wanneer we een klassiek VRP beschouwen zoals hierboven beschreven, dan zijn er enkele zaken die we moeten herzien voor het toevoegen van drones. Allereerst hebben we een nieuw soort voertuig in onze vloot namelijk een drone. Om het verschil tussen drones en wagens duidelijk te maken hebben we ook een notie van snelheid nodig voor wagens v_w en drones v_d. Ook verschilt de capaciteit bij drones c_d waarbij $c_d = 1$ voor drones die dus slechts één pakket per keer kunnen leveren. Door deze restrictie in capaciteit van drones wordt bij VRPD meestal de capaciteit van de locaties op 1 gelaten zoals in het standaard VRP, hierdoor kan elke locatie ook bezocht worden door een drone en kan het volledig potentieel van drones beter getoond worden zonder dat de capaciteit een te groot obstakel vormt. Een VRP dat opgelost wordt met een vloot die bestaat uit verschillende voertuigen met andere eigenschappen zoals in VRPD wordt een VRP met heterogene vloot genoemd.

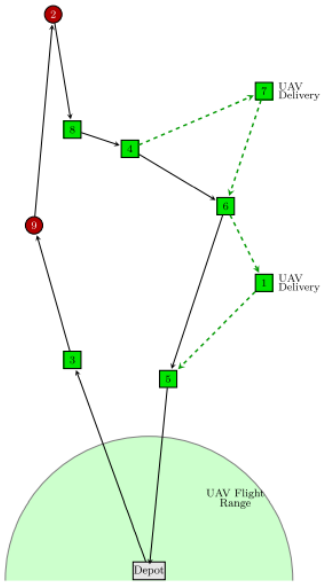
2.2.1 VRP met heterogene vloot

Een belangrijke variant van het VRP doet zich voor wanneer we een vloot hebben die bestaat uit verschillende voertuigen. Deze voertuigen worden gekarakteriseerd door een eigen capaciteit, reis kosten(snelheid), penalty's en andere zaken. Deze variant van het VRP wordt het heterogenous fleet vehicle routing problem (HFVRP) of mixed fleet vehicle routing problem (MFVRP) genoemd, en werd voor het eerst geïntroduceerd in 1984 door Golden et al. [24]. Er bestaan ook hier verschillende sub varianten van dit probleem. Het grootste onderscheid dat dient gemaakt te worden is of we een vloot hebben van vaste grootte, of als we ervan uit mogen gaan dat onze vloot een oneindig aantal voertuigen bevat van de verschillende soorten. Deze laatste variant wordt vaak fleet size mix VRP (FSMVRP) genoemd, deze is vooral interessant om achteraf een vloot samen te stellen die het meest efficiënt kan werken. Bij HFVRP is er een vaste heterogene vloot beschikbaar in een centraal depot die gebruikt wordt om een set van klanten te bedienen. De vloot bestaat uit n verschillende types voertuigen $M = \{m_1, ..., m_n\}$. Voor elk type voertuig is een bepaald aantal beschikbaar in de vloot. Verder heeft elk type vehicle een capaciteit c_i en een reis snelheid v_i, ook andere metrieken zoals kost per uur kunnen beschreven worden per type. Voor verschillende problemen kunnen ook andere karakteristieken verbonden worden aan de types voertuigen, degene die voor ons belangrijk zijn in PDSVRP zijn vooral laadtijd en levertijd. Deze karakteristieken worden vaak als penalty's gemodelleerd die bij elk voorval aangerekend worden. Bij HFVRP wordt een klant bezocht door exact één voertuig, en is het aantal routes met een type voertuig niet groter dan het aantal voertuigen van dat type, dit betekent dus één route per voertuig. Verschillende versies van het probleem laten vaak slechts een aantal verschillen tussen de types voertuigen toe, zoals enkel verschillende capaciteit of reiskosten. Zo bestaan bijvoorbeeld heterogeneous VRP with fixed costs and vehicle dependent routing costs (HVRPFD) waarbij dus penalty's als laden en lossen kunnen verschillen en voertuigen een verschillende transport kost hebben. Een andere variant is heterogeneous VRP with vehicle dependent routing costs (HVRPD) waarbij voertuigen enkel een verschillende transport kost hebben. Er bestaan hier nog tal van andere varianten van met andere sets van contraints.

HFVRP is een vrij complexe vorm van het VRP en wordt daarom ook vaak benaderd opgelost. Alle oplossingsmethoden voorgesteld in de literatuur zijn dan ook methodes met heuristieken en meta-heuristieken, meestal zijn deze methodes extensies van bestaande VRP oplossingsmethoden. Methodes als genetic search [51], tabu search [23], ruin-and-recreate [16] en simulated annealing [40] zijn enkele meta-heuristiek methodes die voor state-of-the-art resultaten hebben kunnen zorgen bij HFVRP.

Bij HFVRP wordt zoals bij cVRP ook de som van de totale afstand van de routes geminimaliseerd voor de gegeven vloot. Dit opnieuw met de contraints dat de som van de capaciteiten van de locaties op een route niet groter mag zijn dan de capaciteit van de wagen die deze route rijdt. En natuurlijk dient opnieuw elke locatie precies eenmaal bezocht te worden.

2.2.2 Wagen met drone side-kick



(b) The UAV is launched from a delivery truck, delivering parcels to two eligible customers.

Figuur 2.1: Voorstelling van het FSTSP (Murray et al. [48])

Het idee van een flying side-kick wordt voor het eerst besproken door Murray and Chu [48]. Ze onderzoeken hoe we drones het beste kunnen toevoegen aan huidige traveling salesman problemen en bespreken en vergelijken daarbij zowel de manier met een drone als side-kick van een wagen, als ook de manier waarbij drones parallel en autonoom van de wagen werken. Beide manieren worden hieronder besproken.

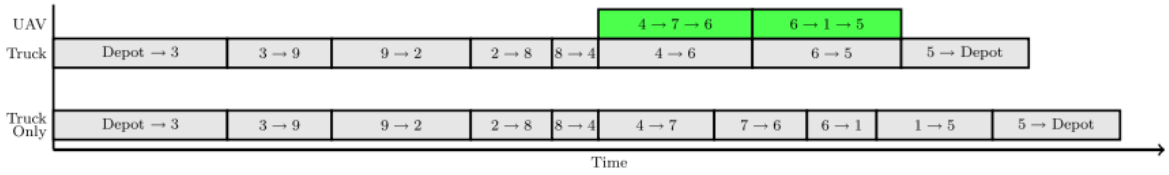
In de paper wordt een drone gebruikt als flying side-kick en stellen ze flying side-kick traveling salesman problem (FSTSP) voor. Dit is een extensie van het traveling salesman probleem waarbij een drone samen met wagens op pad gaat, de drone vertrekt vanaf de wagen en komt hierna terug samen met de wagen. Het grootste voordeel hiervan is dat de range van de drone verlengd wordt door de al afgelegde afstand door de wagen. Hierdoor wordt de beperkte range van een drone optimaal benut. Deze manier van TSP werd door onder andere Wang et al [69] verder ontwikkeld om het compatibel te maken voor VRPs. Bij FSTSP worden de volgende contraints opgesteld:

- een drone kan slechts één locatie per keer bezoeken, en moet hierna een nieuw pakket halen bij de wagen.

- De wagen mag meerdere locaties behandelen terwijl een drone weg is.
- De drone mag enkel opstijgen, en landen tijdens een stop van de wagen.
- Wanneer een drone bij een locatie landt mag deze ook bij dezelfde locatie terug opstijgen, er mag echter niet bij dezelfde locatie eerst opgestegen en erna geland worden.
- Elke locatie wordt exact 1 maal bezocht door een wagen of drone.
- De drone kan zo lang als nodig op de wagen blijven.
- De drones totale afstand kan maximaal de afstand van zijn range constante zijn.

Er wordt echter geen rekening gehouden met de capaciteit van drones en de capaciteit die een locatie nodig heeft, ter vervanging hiervan kunnen sommige locaties zich toegankelijk stellen voor drones en andere met een te hoge capaciteit niet. Een drone kan enkel locaties bezoeken die toegankelijk zijn voor drones.

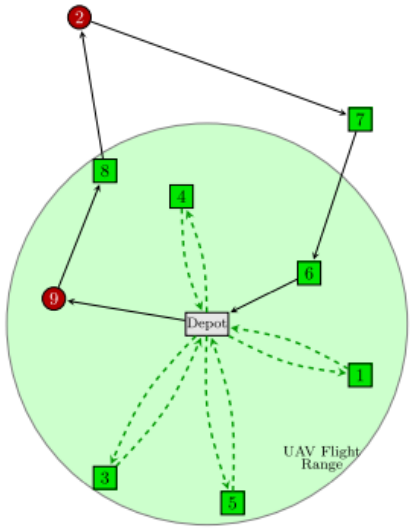
Bij FSTSP wordt de makespan of totale tijd geminimaliseerd waarin de wagen en UAV wegblijven zoals te zien in figuur 2.2.



Figuur 2.2: Vergelijking tussen tijd nodig voor het VRP uit figuur 2.1 uit te voeren met en zonder drone als side-kick (Murray et al. [48])

De grootste zorg in de paper over het gebruik van drones in een TSP is de beperkte range van een drone. De beperking in range is zeer terecht in verschillende situaties, maar in grote steden is dit een minder probleem. Huidige delivery drones hebben al een range van 15-25 km [70] wat genoeg is om in een middelgrote stad zoals Gent alle locaties te kunnen bereiken. Daarboven zal met vele onderzoek dat gedaan wordt naar batterijen en drones deze range ook alleen maar toenemen in de toekomst. De extra technologische ontwikkeling die nodig is om een drone te laten starten en toekomen op een bewegende wagen, samen met het extra werk voor de chauffeur voor het laden en lossen van de drone tijdens de route is ook een grote toegevoegde complexiteit in het lever proces. Potentieel kan dit een te grote stap zijn in het lever proces, aangezien zelfs makkelijke incorporaties van drones in het lever proces nog niet op grote schaal uitgerold zijn. De flying side-kick aanpak van het TSP haalt echter wel het meeste voordeel in landelijke, slecht te bereiken gebieden. Hier zou een parallel drone scheduling TSP zoals hierna besproken een beperkte invloed hebben, veel kleiner dan die van het flying side-kick TSP. In stedelijke gebieden zijn de rollen weer omgedraaid, aangezien de extra laad- en lostijd van de drones door een chauffeur de moeite niet waard is voor de kleine afstanden. Hier dient dus een afweging gemaakt te worden. Maar in theorie zal de flying side-kick methode vaak in minder drukke gebieden beter blijken door de efficiëntere paden die afgelegd kunnen worden door voertuigen en drones, het is echter de toegevoegde complexiteit in het lever proces die dit een onwaarschijnlijke eerste versie van drone delivery maken. Ik ben echter ook van mening dat de theorie hier lastig om te zetten valt in de praktijk, daarom lijkt het uitvoeren van parallel drone scheduling TSP een meer haalbare oplossing om praktisch op grote schaal uit te rollen.

2.2.3 Parallel drone scheduling TSP / standaard VRPD



Figuur 2.3: Voorstelling van het PDSTSP (Murray et al. [48])

Leveringen met drones vanuit het depot is een meer realistische aanpak voor het integreren van drones in het lever proces. Murray et al [48] introduceerde ook het parallel drone scheduling TSP (PDSTSP) in zijn paper over FSTSP (Figuur 2.3). Bij PDSTSP wordt een vloot van drones parallel ingezet, autonoom van een wagen. De start en stop positie van de drones is de depot, in tegenstelling tot de aanpak bij flying side-kick is er dus geen interactie tussen drones en wagens nodig. Wanneer we deze aanpak gebruiken zijn er enkele zaken die we kunnen vereenvoudigen ten opzichte van het side-kick model. Zo bestaat een route uit een aantal locaties die één voor één afgehandeld worden, maar aangezien een drone slechts één locatie per tour kan behandelen zal deze na elke locatie terug naar de depot moeten keren. Het is ook triviaal om te zien dat wanneer een drone een aantal locaties moet behandelen de volgorde waarin het deze locaties bezoekt geen verschil maakt voor de totale tijd. Enkel de verdeling van de locaties onder de drones is belangrijk, zodat deze evenredig gebeurt. Hierdoor kunnen we de locaties van de drones samennemen in een set L_d en het probleem van het inplannen van locaties voor drones omvormen naar een parallel identical parallel machine scheduling probleem (IPS) waarbij we de makespan proberen minimaliseren, en L_d zo gelijk mogelijk verdelen onder de drones. Dit zodat de totale werktijd van het starten van de eerste drone tot het stoppen van de laatste drone geminimaliseerd wordt.

IPS is een klassiek research probleem dat NP-HARD is, er zijn echter al veel methodes gevonden die zeer goede oplossingen kunnen geven in redelijke tijd zoals het genetische algoritme van Min et al [47] een exact algoritme van Xu et al. [73] met column generation. Bij PDSTSP beschreven in Murray and Chu [48] wordt een heuristiek voorgesteld voor de opsplitsing van het probleem in een klassiek VRP en IPS probleem die opgelost worden met een gekende methode. Het (exact) oplossen van een IPS met deze methodes is echter vrij tijdsintensief, en bij vele meta-heuristieken als we dus de fitness van het totale systeem willen bepalen zijn deze te kostelijk.

Een bekende benaderde oplossingsmethode die we als oplossing hiervoor gebruiken is longest-processing-time-first scheduling (LPT) een gretige oplossingsmethode voor dit probleem voor het eerst onderzocht door R. Graham et al. [26]. De werking van dit algoritme gaat als volgt: we sorteren de locaties naargelang hun afstand van de depot (wat overeenkomt met de langste processing time), en schedulen telkens de langste locatie in bij de drone die nog het minste werk heeft. Graham et al.

wist te bewijzen dat deze manier van aanpak in het slechtste geval de maximale processing time een factor $4/3 - 1/(3m)$ zal verschillen van de optimale langste processing time. Hierbij is m het aantal machines waarover het werk verdeeld kan worden of in ons geval het aantal drones. Het bewijs werd later vereenvoudigd door Xiao et al. [72] en wordt hier gegeven.

bewijs Xiao et al. [72]

Normaliseer de processing time t_i van de items I zodat de optimale langste som gelijk is aan 1. Dit betekent dat de som van alle items nu maximaal het aantal machines m zal zijn, dit komt neer op:

$$t'_i = \frac{t_i}{\sum_{j \in I} t_j / m}$$

Deel de items op in 3 groepen, een large groep met items met $t'_i > 2/3$, een medium groep met items met $1/3 \leq t'_i \leq 2/3$, en een small groep met items met $t'_i < 1/3$. Laat het aantal items per groep nu gelijk zijn aan n_L, n_M en n_S In een optimale partitie kan er per machine maximaal 1 item uit de large groep komen, dus $n_L \leq m$. Ook kan een optimale partitie geen large en medium item bevatten, of 3 medium items, hieruit volgt dat $n_M \leq 2(m - n_L)$.

We kunnen de operaties van het gretig algoritme nu opdelen in 3 fases:

1. Alloceer de large items, elk van deze wordt aan een verschillende machine toegewezen. Omdat $n_L \leq m$ heeft elke machine dus hoogstens een large item. Met de maximum som hier gelijk aan 1.

2. Alloceer de medium items. De eerste $m - n_L$ worden bij de lege machines geplaatst.

   - Wanneer er maximaal $m - n_L$ medium items zijn is de som van een machine nog steeds maximaal 1.
   - Wanneer er meer medium items zijn noemen we de grootste $m - n_L$ medium items de top-medium items en de kleinste de bottom-medium items. In het slechtste geval zijn m items allemaal groter zijn dan $1/2$, dan moet in de optimale oplossing deze allemaal bij een andere machine geplaatst worden. Elk bottom-medium item moet dus in de optimale oplossing bij een van deze m grootste items passen. We noemen 2 items matchable als hun som maximaal 1 is, ze kunnen dus samen zitten in een machine in de optimale partitie. Door Hall's theorem, zal elke subset van k bottom-medium items matchable zijn met k van de grootste m items. We hebben dan dat item $\#m$ machable moet zijn met item $\#m + 1$, of over het algemeen dat item $\#m + k$ matchable is met $\#m - k + 1$. Bij LPT doen we exact dit, we plaatsen $\#m + k$ bij dezelfde machine als $\#m - k + 1$ waardoor de som dus telkens maximaal 1 blijft.

3. Alloceer de small items, er zijn hier 2 mogelijkheden:

   - als de huidige kleinste som maximaal $1 - 1/(3m)$ is, dan is de som na het toevoegen van een small item maximaal $1 - 1/(3m) + 1/3 = 4/3 - 1/(3m)$
   - Als dit niet het geval is zijn alle sommen groter dan $1 - 1/(3m)$ en dus is de som van de grootste $m - 1$ langste machines groter dan $(m - 1) \cdot (1 - 1/(3m)) = m - (4/3 - 1/(3m))$. Omdat de som van alle items maximaal m kan zijn, moet de nieuwe som ten hoogste kleiner zijn dan $4/3 - 1/(3m)$.

■

We moeten enkel aangeven dat een locatie door een drone behandeld zal worden om de makespan benaderd te kunnen bepalen. Dit kunnen we doen door de som te nemen van tweemaal de afstand tussen de depot en de locatie, hierna delen we deze som door het aantal drones. Daarna vermenigvuldigen we met de factor $4/3 - 1/(3m)$ aangezien we door het bewijs van Graham et al. [26] gezien hebben dat onze gretige manier van toewijzen maximaal met deze factor verschilt. Hierbij is m het aantal drones.

$$makespan_{drones} \approx (4/3 - 1/(3m)) \cdot \sum_{i=1}^{|L_d|} 2 \cdot dist(l_d, L_d[i])/m$$

De ware kost kan echter nog altijd groter of gelijk zijn aan deze geschatte kost. Dit omdat we nu de ideale kost voor het identical scheduling probleem schatten door de best mogelijke oplossing namelijk dat we in het ideale geval alles perfect kunnen opdelen dat elke drone evenveel werk heeft. Maar door de worst-case van het LPT algoritme te gebruiken cancelt zich dit in de meeste gevallen wel uit waardoor onze approximatie vrij goed is. Het bepalen van de scheduling volgens LPT is echter ook niet zo een computationeel intensieve operatie en kan in een tijds complexiteit van $O(n \log n)$ uitgevoerd worden daarom berekenen we in onze fitness functie de exacte makespan met het LPT algoritme.

Door deze eigenschap kunnen we het klassiek VRP probleem zoals hierboven beschreven vereenvoudigen. De kost om een locatie uit een route weg te halen en aan een drone te geven is namelijk zeer klein. Hierdoor kunnen we en vereenvoudiging van de routes maken die slechts een subset van de locaties behandelt. Op deze manier kunnen locaties die niet efficiënt binnen een route passen overgelaten worden aan drones.
Dit probleem kunnen we zien als een vereenvoudiging van het VRP waarbij de constraint dat alle locaties bezocht moeten worden door de wagen wegvalt. De overgelaten open locaties worden aan drones toegewezen en met LPT ingepland. In de literatuur valt dit probleem onder de noemer van de selective traveling salesman problems (STSP) [36] of TSPs with profits [21]. Dit omdat er vaak bij dit soort TSPs elke customer een profit geeft en een vehicle een bepaalde capaciteit of tijdslimiet heeft, het doel is om de winst te maximaliseren zonder de capaciteit/limiet te overschrijden. Ons specifieke probleem waarbij we meerdere routes hebben en de routes een subset van de klanten bezoekt wordt besproken onder 2 noemers, waarvan de bekendste het team orienteering problem (TOP) [9], maar het wordt soms ook het profitable tour problem (PTP) [21] genoemd. Er is echter een klein verschil tussen beide problemen. Bij het team orienteering problem is de objective function waarnaar we optimaliseren de totale winst met een beperkte capaciteit/tijd. Bij profitable tour problem optimaliseren we echter naar een combinatie van de totale winst en de reiskost. In ons geval is de limiterende factor de tijd die een koerier heeft en de capaciteit van een vehicle, ook is elke customer even profitable. We kunnen echter ook een reis kost afleiden op basis van de afstand. Daarom kunnen we zowel TOP als PTP gebruiken als vereenvoudiging van het VRP.

Als we het TOP/PTP oplossen kunnen we de overgebleven nodes, die niet toegewezen zijn aan een route, toewijzen aan drones en bekomen we een goede oplossing. Aangezien de kost van de drones zo snel benadert te berekenen valt kunnen we deze makkelijk ook direct meegeven in de totale fitness functie waardoor we naar zowel het vereenvoudigd VRP als de drone delivery kunnen optimaliseren, en dus de kost van het volledige systeem minimaliseren in onze meta-heuristiek. Deze nieuwe eigenschap van VRP met drones maakt het mogelijk om veel mutaties te doen op routes die anders zeer inefficiënt waren. Zo kunnen we bijvoorbeeld zonder een grote kost destruct-and-repair toepassen. Waarbij we een inefficiënt stuk in onze routes volledig afbreken en opnieuw toewijzen, daarbij kan nu voor zo efficiënt mogelijke routes gegaan worden en het overschot van locaties kunnen we aan de drones toekennen. Bij een klassiek VRP moet bij destruct-and-repair alle locaties aan andere routes toegekend worden wat vaak zeer moeilijk kan zijn om snel te doen en bekomt vaak inefficiënte routes. We kunnen nu ook constructie van routes op een natuurlijke manier laten verlopen door goede locaties op te nemen in routes en slechte open te laten voor andere routes zonder deze direct aan een andere route toe te moeten wijzen. Bij VRPs zijn dit soort operaties ook mogelijk, maar nadien dient nog een repair

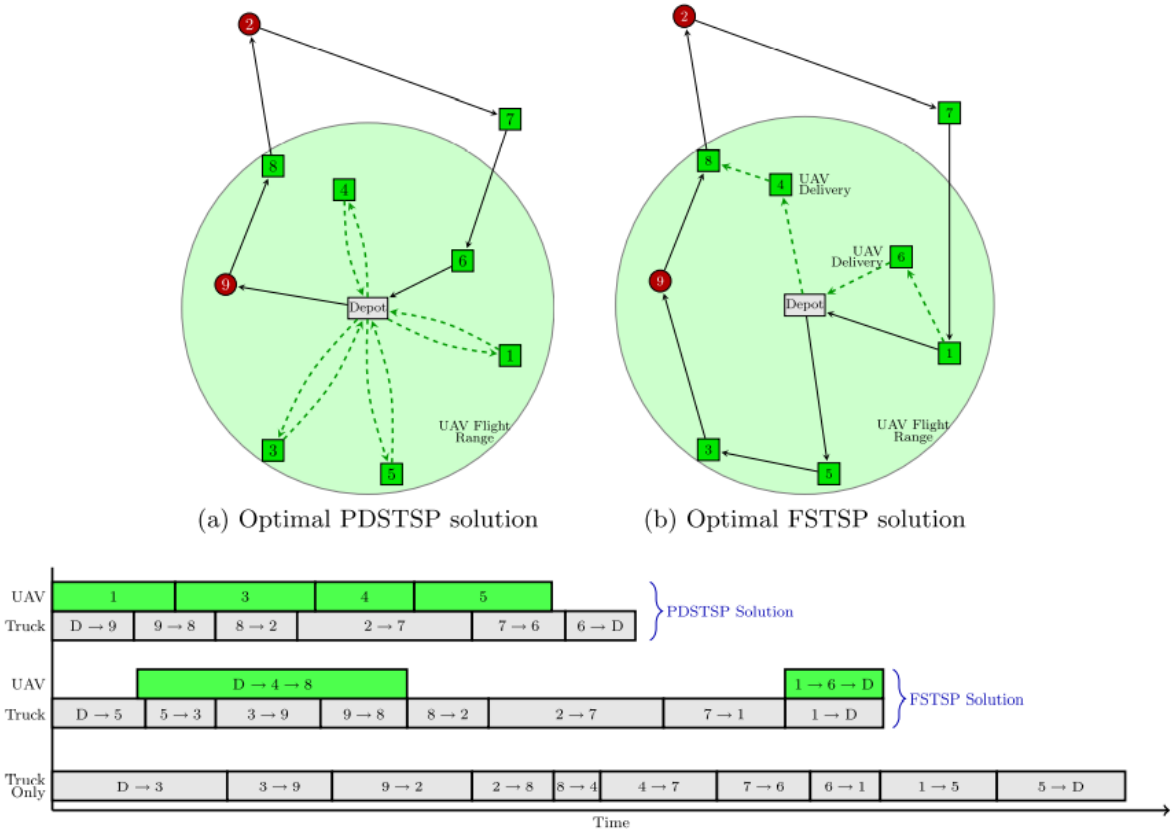
operatie te gebeuren die alle open locaties toekent, wat vaak zeer kostelijk is en voor slechte resultaten kan zorgen.

Parallel drones scheduling multiple traveling salesman problem (PDSmTSP)

PDSmTSP is zeer gelijkaardig aan het net besproken PDSTSP het enige verschil is dat we bij een multiple traveling salesman problem (mTSP) meerdere routes construeren. Het verschil tussen een VRP en een mTSP is dat bij mTSP er geen rekening gehouden wordt met de capaciteit van het voertuig. mTSP is dus een vereenvoudiging van het VRP. Wanneer de capaciteit van het voertuig geen limiet vormt bij een VRP dan is het probleem equivalent aan een mTSP. Bij mTSP kunnen we m voertuigen gebruiken, als we deze m gelijkstellen aan 1 dan bekomen we het klassieke TSP, hierdoor is het dus een generalisatie van het TSP. Bij mTSP is er ook slechts 1 depot en minimaliseren we de kost. Hierdoor kunnen alle methoden ontwikkeld voor TSP dus ook makkelijk gebruikt worden voor mTSP, en de methodes voor VRP ook gebruikt worden voor mTSP. Elke oplossing van een VRP is ook een oplossing voor een mTSP, maar niet andersom.

Door deze eigenschappen kunnen we de algoritmes die we ontwikkelen dus ook gebruiken voor PDSmTSP. En hierdoor hebben we een extra klasse problemen waarop we ons algoritme kunnen testen. De eerste, en voorlopig enige paper die ik gevonden heb over PDSmTSP is zeer recentelijk uitgekomen door Salue et al. [58]. We bespreken ook uitvoerig de methodiek en oplossingen van deze paper en vergelijken deze met onze bekomen oplossingen.

2.2.4 Vergelijking tussen FSTSP en PDSTSP



(a) Optimal PDSTSP solution

(b) Optimal FSTSP solution

(c) A comparison of delivery schedules for the two systems depicted above. The "Truck Only" schedule represents an optimal TSP solution for a single truck serving all customers.

Figuur 2.4: Vergelijking tussen de makespan van PDSTSP en FSTSP (Murray et al. [48])

Het grootste verschil tussen FSTSP en PDSTSP op basis van hun eigenschappen is vooral dat FSTSP veel minder afhankelijk is van de range van de drone. Bij PDSTSP zijn onze leveringen door drones gelimiteerd tot afstanden die binnen de range van de drone vallen. Bij FSTSP valt deze restrictie weg en kan een drone dus ook toegepast worden verder van een depot. Als we de makespan vergelijken in een geval waar de meeste locaties binnen de drone range liggen dan zien we dat de PDSTSP beter zal presteren aangezien de drones veel meer trips zullen kunnen doen in vergelijking met drones als sidekick van een wagen die nog steeds beperkt is tot 1 batterij per trip. Zeker bij lange routes van wagens valt dit makkelijk in te zien aangezien de drone dan voor het grootste deel stil zal moeten zitten op de wagen omdat de batterij (bijna) op is. Maar wanneer we kijken naar de totale afstand, zien we dat dit significant lager zal zijn bij de FSTSP en zelfs het laagste wanneer we enkel wagens gebruiken. Ook dit is makkelijk in te zien aangezien wanneer een drone opstijgt een locatie bezoekt en terug komt hebben we een driehoek gemaakt waarvan de drone 2 zijden heeft afgelegd naar een punt en de wagen de andere zijde gereden heeft naar het volgende punt, we hebben nu 3 zijden van de driehoek die als afstand meetellen terwijl door 2 zijden te doen met de wagen ook alle 3 de locaties op de punten bezocht hebben, we kunnen dit ook analoog uitbreiden met meerdere locaties tussen het drone vertrek en aankomstpunt. Wanneer we echter de kost van het geheel willen minimaliseren kijken we gezien de vaste loonkost van een koerier best naar de laagste makespan. In dit geval zien we ook vaak dat PDSTSP de bovenhand neemt wanneer de meeste locaties binnen de range vallen. Zoals

eerder besproken is de toegevoegde complexiteit van FSTSP aan het lever proces ook een veel grotere stap. Daarom gaan we in deze scriptie enkel verder met het ontwikkelen van PDSTSP voor VRPs en dynamische VRPs.

2.3 Dynamic vehicle routing problem (DVRP)

Een van de hoofdredenen waarom we drones zouden inzetten in de transportsector is om de efficiëntie van de leveringen te bevorderen. Het gaat vooral om de levertijden inkorten, de kost reduceren en mankracht verminderen. Het ideaal doel dat we hierbij voor ogen hebben is een systeem dat same-day delivery toelaat. Om echter same-day delivery mogelijk te kunnen maken moeten er ook voortdurend locaties kunnen toegevoegd worden aan de berekeningen. Dit aangezien er de dag zelf nog bestellingen bijkomen terwijl de berekeningen aan het lopen zijn. Een groot verschil bij DVRP is ook dat er meerdere routes per koerier gereden kunnen worden. We dienen dus op elk moment van deze berekeningen een zo goed mogelijke route te kunnen opvragen. Dit omdat op elk moment een koerier kan terugkomen van zijn route en we hem een nieuwe route moeten kunnen toewijzen. Deze dynamische vorm van VRP noemen we het dynamisch VRP (DVRP).

Wanneer we in de literatuur dynamic VRP bekijken komen we vaak op het probleem waar de routes zelf nog dynamisch aangepast worden tijdens het uitvoeren ervan [37]. Dit is zo in het traveling repairman problem [7], waar dynamisch nog aanpassingen kunnen gemaakt worden aan de route tijdens het rijden ervan. Maar dit is niet het dynamische VRP dat wij behandelen, in het geval van het leveren van pakjes gaat dit echter niet omdat de pakketten uniek zijn en toekomen in de centrale. Daarom praten we hier over dynamisch in de zin dat het probleem continu aangepast wordt tijdens de berekeningen, maar eens een route gestart wordt zal deze niet meer aangepast worden en is deze definitief. Dit is hetzelfde probleem zoals behandeld wordt in de paper van Ichoua et al. [30]. In het dynamic vehicle routing problem (DVRP) dat wij dus behandelen, hebben we een set van locaties L die niet langer constant is zoals bij het statische VRP. Tijdens de berekeningen kan een locatie l_{n+1} toegevoegd worden aan de set van locaties. Ook kunnen locaties uit de berekeningen verwijderd worden, bij het rijden van een route door een drone of wagen zullen de locaties van deze route uit de berekeningen gehaald worden. Deze staan namelijk vast om geleverd te worden en maken dus geen deel meer uit van het probleem. Deze extra contraints brengen enkele nieuwe uitdagingen met zich mee, zo moet er een manier komen om efficiënt een locatie toe te voegen aan een oplossing, maar moet vooral ook opnieuw nagedacht worden over hetgeen waarnaar we optimaliseren. We kunnen nog steeds naar gelijke metrieken optimaliseren zoals totale afstand of efficiëntie. Maar zeker efficiëntie wordt nu een pak interessanter. Aangezien het probleem nu kan veranderen kunnen locaties die niet efficiënt op een route passen mogelijks later wel efficiënt op een andere route terecht komen waardoor het soms niet de moeite is de locatie op te nemen in een route. Ook zullen metrieken die goed werken op statische problemen zoals afstand ook inefficiënte locaties op de routes moeten plaatsen omdat deze opgenomen moeten worden. Maar bij een dynamische vorm met meerdere routes kunnen we wachten tot het gunstiger wordt deze inefficiënte locaties te doen. In de praktijk is dit ook een logische stap die gezet zal worden. Wanneer we bijvoorbeeld een volledige stad dienen te behandelen in 4 shiften, is het logisch dat we niet elke shift de afgelegen delen van de stad zullen bezoeken, maar slechts eenmaal deze bezoeken als er ook hier al enkele locaties opgehoopt zijn.

In de literatuur wordt voor DVRP traditioneel de fitness functie van Ichoua et al. [30] gebruikt. Deze minimaliseert de totale kost gemeten door de som van de totale afstand en de totale lateness over alle locaties. Deze lateness is een penalty voor hoeveel te laat een pak geleverd werd. Maar aangezien we niet met een time-widow of levertijd werken is dit bij ons niet echt van toepassing.

2.3.1 Same-day delivery

Als we nadenken aan de volgende stappen voor de transportindustrie, merken we dat snellere leveringen en goedkopere leveringen prioriteit krijgen. Het lijkt dus onvermijdelijk dat transportbedrijven hun operaties nog meer zullen moeten optimaliseren om snellere en goedkopere leveringen te kunnen garanderen. Momenteel bieden de meeste leverancier bedrijven al next-day delivery aan. Als deze evolutie in leveringen zich doorzet lijkt same-day delivery dus onvermijdelijk. De grootste uitdaging bij same-day delivery is echter dat de efficiëntie van de leveringen noodgedwongen achteruit zal gaan. Wanneer we echter een volledig beeld hebben aan de start van de dag van alle leveringen die moeten gebeuren die dag, is het logisch dat we betere beslissingen kunnen maken over de routes die zullen moeten gereden worden. We zitten bij SDD en DVRP dus met incomplete informatie en een graad van onzekerheid. Ook kunnen we zoveel mogelijk pakketten telkens meegeven aan de koerier waardoor in een route meer locaties behandeld kunnen worden. Minder pakketten per route betekent over het algemeen ook lagere efficiëntie. Door de continue toestroom van pakketten en beschikbaarheid van slechts een deel van de pakketten op een bepaald tijdstip, is het noodzakelijk dat koeriers meerdere routes doen vanuit de depot. Het is triviaal dat dit de efficiëntie vermindert aangezien er meer laadtijd komt, maar ook geen grote routes met veel locaties gereden kunnen worden. Het is ook noodzakelijk dat we een efficiënt online algoritme ontwikkelen om deze routes in te plannen en tijdens de berekeningen aan te passen door nieuwe locaties te introduceren of routes te verwijderen wanneer deze gereden worden. We hebben dus een dynamisch VRP nodig en een efficiënte oplossingsmethode wanneer we SDD willen faciliteren.

2.4 Dynamic vehicle routing problem met drones (DVRPD)

Een mogelijke oplossing voor het verder optimaliseren van en versnellen van leveringen is het introduceren van unmanned vehicles. Nu zit de bottleneck van de leveringen ook bij de mankracht die ontbreekt. Drones vormen hier een oplossing die op een toegankelijke kostenefficiënte manier unmanned delivery kan mogelijk maken. Daarnaast lenen drones zich ook zeer gemakkelijk aan het dynamische model wat nodig is voor SDD, dit omdat drones na elke levering terug in de depot komen en er dus ruimte is om aanpassingen te maken na elke levering. Dit is niet mogelijk bij wagens aangezien deze de pakketten voor de locatie bij vertrek van de depot moeten meenemen. Vanaf een route gereden wordt is deze ook definitief en wordt deze ook uit de online berekeningen verwijderd.

Er zijn reeds oplossingsmethoden ontwikkeld voor het probleem dat we net besproken hebben. De meest besproken oplossing werd voorgesteld in Ulmer et al [62]. Deze oplossing bestond echter uit een manier om het probleem op te splitsen in een standaard VRP met wagen en een VRP met enkel drones. Zijn voorgestelde methode SDDPHF gaat hij bij het toekomen van nieuwe locaties met behulp van een techniek genaamd parametric policy function approximation beslissen of deze locaties door drones of wagens verwerkt zal worden. Hierna worden standaardmethoden voor VRPs toegepast om de routes en toewijzingen te berekenen. Deze methode is een degelijke oplossing voor het probleem, aangezien we de best gekende VRP solvers in kunnen pluggen. Maar er bestaat nog steeds een inefficiëntie door het vast toewijzen van de locaties. Het probleem wordt er opgedeeld in 2 afzonderlijke problemen die elk op zich geoptimaliseerd worden. Er is echter meer winst uit het systeem te halen als we beide problemen samen proberen optimaliseren en locaties tussen de problemen doorgegeven kunnen worden. Zeker omdat we hier met een dynamisch probleem zitten kan het mogelijks zijn dat door een verandering in de situatie een beslissing uiteindelijk niet ideaal blijkt te zijn. Daarom is er hier gekozen een nieuwe oplossingsmethode te ontwikkelen die tegelijk de routes construeert en de locaties toewijst, en dan een minimum van het volledige systeem probeert te zoeken. Hierdoor kunnen dynamische aanpassingen in de situatie direct opgenomen worden in de berekeningen. Zo kunnen we de routes en aanwijzing van locaties zich beter laten aanpassen aan de continu veranderende situatie. Op die manier kunnen we ook de nieuwe mutaties en strategieën toepassen die mogelijk zijn in onze vereenvoudigde vorm van het VRP besproken in VRPD.

Bij DVRPD wordt ook zoals bij DVRP meerdere routes gereden door een koerier, we hebben

hierdoor dus een nieuwe eind constraint nodig zoals bij DVRP. In DVRP wordt vaak nog steeds de totale afstand gebruikt in combinatie met een lateness penalty, maar wij stellen een leveringen/uur fitness functie voor die we willen maximaliseren. Onze uiteindelijke doel-functie is zoveel mogelijk pakjes geleverd te hebben met SDD net zoals bij Ulmer et al. [62]. Hierbij werken we dus met een vaste duur waarin we het aantal geleverde pakjes willen maximaliseren. De constraint dat elk pakje geleverd kan worden kan dan ook niet altijd meer voldaan worden.

2.5 PDSVRP/DVRPD fitness functies

Een belangrijk onderdeel bij het probleem dat we oplossen, en de definitie van het probleem zelf, is waarnaar we proberen optimaliseren. In ons geval is hier geen triviaal antwoord op en vaak wordt in de literatuur ook vaag gesproken over het minimaliseren van de kost van het systeem. Zeker bij dynamische algoritmes is het soms lastig een kost te definiëren aangezien we altijd slechts een snapshot van het geheel hebben op een bepaald tijdstip.

2.5.1 Minimaliseren van de totale afstand

Bij het standaard cVRP is deze kost voornamelijk de som van de afstanden van de routes die de capacity contraint niet overschrijdt. Deze capacity contraint wordt als volgt gedefinieerd:

$$\forall r \in routes : capacity(truck) \leq \sum_{l \epsilon r} q_l$$

Als deze voldaan is berekenen we de fitness als volgt:

$$fitness = \sum_{r \epsilon routes} dist_r$$

Hierbij wordt dus geen rekening gehouden met de tijd van de route, of het aantal locaties ervan of zelfs niet de verdeling van het werk onder de koeriers. Zolang alles binnen de capacity contraint valt is het in orde, en minimaliseren we de afstand. Hierdoor kunnen we echter ook vaak routes krijgen die niet ideaal zijn in een praktisch scenario. Grote routes worden hierdoor namelijk vaak verkozen aangezien we bij routes vaak een initiële afstand hebben naar een bepaalde omgeving waar het voertuig alle locaties behandeld en erna terug keert. Als we de routes zo lang mogelijk maken hebben we minder deze initiële lange afstand. Maar hierdoor kunnen routes soms overblijven met zeer weinig locaties over. Hierdoor krijgen we enkele zeer lange routes ver weg, en enkele zeer korte routes kortbij, zo hebben we de lange afstand naar de verre routes een pak minder moeten doen, maar is het werk veel minder goed verdeeld en zullen de verre routes veel langer zijn dan de korte. Zeker als er een groot verschil is tussen de totale capaciteit van een vloot en de capaciteit van de te leveren pakjes zien we slecht verdeelde routes.

2.5.2 Minimaliseren van de totale tijd

Het gebruiken van de totale afstand als fitness metriek is echter niet nuttig voor PDSVRP of DVRPD, dit aangezien de totale afstand gereden door een wagen steeds kleiner of gelijk aan die van een drone zal zijn. Een belangrijke kanttekening die we hier moeten maken, is dat we bij het VRP de afstand tussen twee locaties als Euclidische afstand tussen de 2 coördinaten nemen, in een praktisch leverscenario zal een drone wel in vogelvlucht kunnen vliegen en een kortere afstand tussen 2 locaties moeten afleggen. Aangezien een drone telkens terug moet keren naar de depot, kan de afstand niet korter dan een wagen. Het is makkelijk in te zien met behulp van de driehoeksongelijkheid, dat een route door een drone te laten behandelen altijd een langere totale afstand tot gevolg zal hebben. Dit kunnen we als volgt bewijzen:

bewijs

Stel we hebben een oplossing met een route r van een wagen en een locatie L die door een drone behandeld wordt. Wanneer de wagen genoeg capaciteit heeft kunnen we ook de locatie L invoegen aan het einde van de route na de laatste locatie L_e. De afstand afgelegd door de drone is $2 \cdot d(L, L_0)$ en de afstand van de route is d_r, de afstand van de route tot en met L_e is d_{re}. De totale afstand is dus te schrijven als $d_{re} + d(L_e, L_0) + 2 \cdot d(L, L_0)$. Wanneer we L zouden toevoegen na L_e aan r dan wordt de totale afstand $d_{re} + d(L_e, L) + d(L, L_0)$. Dit zal dus een kortere afstand geven wanneer $d(L_e, L) \leq d(L_e, L_0) + d(L, L_0)$, door de driehoek ongelijkheid kunnen we besluiten dat dit altijd het geval zal zijn. ■

Wanneer we drones introduceren zal de som van de afstanden afgelegd door de voertuigen dus enkel toenemen. Ditzelfde probleem is er ook bij andere HFVRP waarbij verschillende voertuigen ook andere capaciteiten hebben. In benchmark datasets zoals Golden et al. [24] wordt ook de som van de reis kost geminimaliseerd, maar met een vaste vloot. Hierdoor worden vaak ook de s met grote capaciteit intensiever gebruikt, maar dit is ook deel van het probleem. Hier is er wel geen sprake dat wagen A altijd een kortere afstand zal hebben dan wagen B, wat bij voertuigen en drones wel het geval is. Mochten we het op dezelfde manier als HFVRP benchmarks optimaliseren dan zouden drones enkel gebruikt worden als de capaciteit van alle wagens vol is, en er dus geen andere keuze is dan om drones te gebruiken. Hierdoor moeten we dus afstappen van het minimaliseren van de totale afstand en andere alternatieven zoeken.

De manieren waarop drones uitblinken is vooral in hun lage operatie kost, geen restrictie in werkuren, snelheid en het feit dat ze geen last hebben van verkeer en in vogelvlucht kunnen vliegen. Drones kunnen dus geen kortere afstand hebben in het VRP, maar kunnen de uiteindelijke kost wel serieus verminderen. Een mogelijke optie om een nuttige metriek te hebben is de som van de tijd minimaliseren. Aangezien drones sneller kunnen vliegen, laden en lossen kan het wel gebeuren dat deze sneller zijn en een tijdswinst kunnen opleveren. Om echter de tijd van een route te berekenen hebben we wel snelheid en tijd penalty's voor laden en lossen nodig. Maar zelfs met deze metriek minimaliseren we de kost nog niet volledig, aangezien de kost per uur kan verschillen voor wagens ten opzichte van drones. We kunnen een generieker model maken met een reis kost en een penalty kost voor laden en lossen, of een kost per voertuig per uur.

2.5.3 Minimaliseren van de makespan

Een andere mogelijkheid die we hebben als doel-functie is het minimaliseren van de makespan, of de tijd waarin we alle locaties bezocht hebben van begin tot einde. Dit komt dus neer op het minimaliseren van de langste route, en de lengte van een shift dus zo klein mogelijk maken. Voor praktische scenario's lijkt dit een zeer logische metriek aangezien we graag het werk zo gelijk mogelijk verdelen en zo snel mogelijk klaar willen zijn met het werk. Met deze metriek zullen deze twee dingen geoptimaliseerd worden. In ons geval hebben we echter ook locaties die we openlaten en toewijzen aan drones. Hierdoor zou bij het minimaliseren van een maximale route alle locaties opengelaten kunnen worden waardoor de afstand van de routes 0 wordt, en de drones alle locaties moeten behandelen. We dienen dus een uitbreiding te maken waarbij we de afstand van de drones ook in rekening brengen. We kunnen de maximale tijd voor een wagens en drones minimaliseren. We moeten in dat geval de routes van een drone uitbreiden van een locatie per keer te bekijken naar het bekijken van de totale tijd om alle locaties met de drones te behandelen. We berekenen de totale tijd voor drones als volgt: we wijzen de vrije locaties L_{free} toe aan drones en berekenen de totale tijd met:

$$t_{drones} = \sum_{l \epsilon L_{free}} 2 \cdot dist(l_0, l)/v_{drone} + t_{load} + t_{deliver}$$

Dit is de tijd nodig om alle pakketten te leveren met alle drones. Als we de tijd willen hebben voor 1 drone moeten we deze tijd dus nog delen door het aantal drones. Zoals besproken bij PDSVRP, is het werk echter niet exact op te delen en is het opdelen ervan een identical machine scheduling problem. Daarom nemen we de worst-case zoals bewezen in het bewijs van Graham et al. [26] van 4/3 - 1/3m. De tijd voor een drone wordt het volgende:

$$t_{drone} = \frac{(4/3 - 1/3m) \cdot t_{drones}}{n_{drones}}$$

Voor een route is de tijd de som van de reistijden tussen de locaties en de penalty's voor het laden en lossen.

$$t_r = t_{load} + \sum_{i=0}^{|r|-1} dist(r[i], r[i+1])/v_{truck} + t_{deliver}$$

De totale fitness wordt gegeven door het maximum van alle tijden.

$$fitness = \max\{t_r \in routes, t_{drone}\}$$

Dit is een goede aanpak en een fitness functie die degelijke resultaten zal geven, maar er zijn enkele zeer belangrijke minpunten van deze fitness functie. Deze zal het probleem vrij statisch evalueren, en optimaliseert vooral het huidige probleem, zonder de toekomst in acht te nemen. We kunnen echter een fitness functie definiëren die meer inspeelt op het dynamische aspect van het probleem, hierdoor is het minder geschikt voor een dynamisch probleem als DVRPD. Een ander veel groter probleem is dat bij deze fitness functie enkel verbeteringen aan de slechtste route een verbetering zullen geven aan de totale fitness score. Wanneer we een verbetering maken aan andere routes dan de slechtste, zal de fitness hierdoor niet dalen. Dit is dus een groot probleem voor een fitness functie van een meta-heuristiek aangezien we elke verbetering willen meenemen en enkel verbeteringen van de beste route meenemen voor een zeer trage verbetering zal zorgen. Ook al is dit soms de metriek waarnaar we willen optimaliseren in de praktijk, kan dit niet efficiënt gebruikt worden in een meta heuristiek, en dienen we alternatieven te zoeken.

2.5.4 Maximaliseren van leveringen/uur

De besproken fitness functies gaven vaak een score over de oplossing voor het volledige probleem, maar in de dynamische setting met meerdere routes is het niet belangrijk dat we alle locaties doorlopen in een route aangezien we toch meerdere routes zullen doen per koerier. We hebben een metriek nodig die ons kan vertellen welke de beste pakjes zijn die we kunnen behandelen en hoe efficiënt de route is. Op deze manier maakt het niet uit dat de routes alle pakjes behandelen, maar kijken we meer naar welke pakjes kunnen we nu het beste leveren en welke laten we beste wachten voor een volgende route met als doel zo efficiënt mogelijk alle pakjes te leveren. Stel bijvoorbeeld in de praktijk dat we SDD aanbieden, dan is de kans groot dat we niet alle pakjes met SDD zullen kunnen leveren, maar dan willen we graag het aantal pakjes geleverd met SDD maximaliseren. We kunnen dit online doen door ons aantal pakjes geleverd per uur te maximaliseren op elk tijdstip. Op deze manier houden we het dynamische aspect in acht door de efficiëntie van het leveren van de pakjes te gebruiken. Zo kunnen we een nieuwe metriek definiëren die de efficiëntie van de levering bepaald doorheen de tijd. Op deze manier maken we meer gebruik van het dynamische aspect en gaan we telkens vooral de efficiënte pakjes leveren, en geven we meer tijd aan minder efficiënte stukken. Hierdoor proberen we zoveel mogelijk pakjes geleverd te hebben op het einde van de dag. Eigenlijk is deze manier van aanpakken vrij natuurlijk, als we een voorbeeld nemen van een stad als Gent, dan is het logisch dat we vaker routes zullen rijden doorheen het centrum waar veel pakjes besteld worden, maar ook na een tijd meer de stadsrand ook zullen doen als er zich daar ook pakjes beginnen opstapelen. Er is echter nog een probleem met de huidige aanpak, als we geen kost rekenen voor het laden en lossen van de koerier zullen hier kleine routes die zeer efficiënt in elkaar zitten naar voor geschoven worden. We hebben dus een penalty nodig voor de

laad en los kost om grote routes als efficiëntste naar voor te laten komen. Maar zelfs met realistische penalty's voor laad en los kosten zullen soms kleine routes verkozen worden, en hierdoor krijgen we het fenomeen waar onze efficiëntie eerst zeer hoog is, maar elke nieuwe route sterk afneemt. Daarom kunnen we een extra penalty geven t_min die een minimale tijd penalty is waardoor de tijd toeneemt en de routes makkelijker minder efficiënte routes ook kunnen opnemen met nog steeds een toename in leveringen/uur.

De leveringen/uur berekenen we als volgt voor een route r:

$$fitness_r = \frac{|r|}{t_{min} + t_r}$$

Hierbij is $fitness_r$ de leveringen/uur voor route r en is t_{min} een penalty voor de minimale tijd van een route. Als we de werkelijke metriek willen moeten we de t_{min} penalty gelijkstellen aan 0, maar dit is eigenlijk ook een afweging die gemaakt dient te worden want wanneer we t_{min} verhogen zullen onze routes groter worden, maar zal de efficiëntie ervan afnemen. Bij een kleine t_{min} zullen we kleine routes bevoordelen die een goede efficiëntie hebben, maar hierdoor zal onze efficiëntie over een lange periode over het algemeen elke trip verminderen, hierdoor kan het vaak beter geweest zijn om toch een grotere route te pakken. Dit heeft ook veel te maken met hoe frequent locaties toegevoegd worden, als er namelijk snel locaties bijkomen kan steeds kleine routes goede resultaten blijven geven, maar als we na verloop van tijd enkel de minst efficiënte nodes overhouden kunnen onze laatste routes enorm inefficiënt worden. Maar het is een goede zaak dat de lengte van de routes met deze parameter wat geregeld kan worden.

Voor drones is er ook een aanpassing nodig in de fitness functie en dient ook een extra afweging gemaakt te worden in de tijd. Als we namelijk alle open locaties nog steeds beschouwen als drone locaties zullen vaak onze drones zeer inefficiënt lijken aangezien veel inefficiënte locaties van routes opengelaten worden. Daarom is het logischer een subset te nemen van een bepaalde tijdspanne die we aanwijzen aan drones, de andere locaties laten we open. We kunnen bijvoorbeeld voor 1 uur bepalen hoeveel pakketten de drone kan leveren, en zo de pakketten/uur voor de drone bepalen. Het is triviaal in te zien dat telkens de dichtste open locatie toewijzen aan een drone voor het meeste leveringen/uur voor drones zal zorgen. We dienen dus de open locaties te sorteren volgens hun locatie ten opzichte van de depot, daarna kunnen we de pakketten per uur makkelijk bepalen met algoritme 1.

We berekenen dus de pakketten per uur voor drones als volgt:

Algorithm 1 fitness berekening voor drones

```
sl ← sorted free locations by distance to depot
drones ← amount of drones
totalTime ← 0
parcels ← 0
for l in sl do
    totalTime ← totalTime+ dist(l, depot) / v_d
    parcels ← parcels + 1
    if drones · t_drone < totalTime then
        return parcels/totalTime
    end if
end for
return parcels/totalTime
```

Hierbij is t_{drone} nu gelijk aan de tijd waarvoor we de leveringen/uur van de drones bepalen. De beste waarden voor t_{drone} is gelijk aan de gemiddelde of maximale tijd dat de wagens weg zullen blijven. Zo zal bij de nieuwe routes telkens een goede inschatting komen van het aantal leveringen/uur voor de volgende reeks van routes. Let op hierdoor krijgen we de fitness voor alle drones samen. Aangezien deze verdeling niet perfect kan gebeuren is dit slechts een benadering. Voor de fitness van het geheel

sommeren we de fitness scores van de drones en de voertuigen dit aangezien we de pakketten/uur van het volledige systeem proberen optimaliseren.

$$fitness = fitness_{drones} + \sum_{r \epsilon routes} fitness_r$$

leveringen/uur in het dynamisch vs statisch geval

Wanneer we in de dynamische setting zitten bij een variant van VRPD kunnen we enkel locaties van de vrije locaties openlaten en niet meetellen in de fitness functie aangezien er geen constraint is die zegt dat we alle locaties moeten bezoeken. Deze locatie worden niet behandeld door drones of wagens in de tour. Deze dienen later geleverd te worden. In de dynamische setting hopen we inefficiënte locaties/gebieden over te slaan in de hoop dat we deze later efficiënter kunnen opnemen in een efficiëntere route. Wanneer we d_{time} verminderen in algoritme 1, en t_{min} verlagen in de formule van de fitness laten we toe meer pakjes open te laten. Dit kan gunstig zijn als we een hoge frequentie hebben aan pakjes die erbij komen. Zo kunnen we bijvoorbeeld voor de spits de $t_m in$ gelijkstellen aan de tijd na de spits en kunnen we eerst een vrij kleine route doen om erna met alle nieuwe pakjes van de spits een grote efficiëntere route te maken.

Wanneer we echter in de statische setting werken is leveringen/uur optimaliseren ook een zeer goede metriek, dan moeten we echter terug alle pakjes in de routes opnemen. Dit is makkelijk gedaan door in algoritme 1 bij het berekenen van de drone fitness niet meer te stoppen wanneer $drones \cdot t_{drone} < totalTime$, maar altijd alle vrije locaties op te nemen door de drones. Hierdoor zullen altijd alle locaties opgenomen worden in de fitness functie en zal het systeem dus altijd het volledige VRP beschouwen en hiernaar optimaliseren.
Het leveringen/uur algoritme is dus vrij generiek en kan met het aanpassen van een paar parameters omgevormd worden van een fitness functie voor statische VRP naar dynamische VRP, in onze reference oplossing gebruiken we het echter enkel voor de dynamische setting omdat deze hiervoor het meest geschikt is en bij statisch de makespan een betere keuze is die ook voornamelijk in de literatuur gebruikt wordt.

2.5.5 Minimaliseren van de logistieke kost

In veel verschillende papers over heterogene VRPs en vooral ook VRPD worden effectieve logistieke kosten geschat. Bijvoorbeeld bij Wang et al [69] worden de logistieke kosten van het VRP berekend en geminimaliseerd. Hierbij is een vaste uurlijkse kost voor het operationeel houden van een koerier gerekend, hieronder vallen dingen zoals loon van de koerier. En hiernaast is er ook een reis kost voor drones en wagens waaronder slijtage en energie kosten vallen. Dit is natuurlijk slechts een benadering. De formule voor het berekenen van de kost is de volgende:

$$time_r = dist_r / v_w + t_{load} + |r| \cdot t_{deliver}$$

$$time_d = \frac{\sum_{l \epsilon L_d} dist(l_0, l)/v_d + t_{load} + t_{deliver}}{drones} / v_d$$

$$Cost_{tot} = max(time_r) \cdot Cost_w + drones \cdot time_d \cdot T_d + \sum_{r \epsilon routes} time_r \cdot T_w$$

Hierbij zijn T_w en T_d de kost per uur voor het operationeel houden van wagens en drones. $Cost_w$ is de loon kost die voor iedereen elke dag dezelfde is, aangezien iedereen gelijke werkuren heeft zal dit voor elke persoon dezelfde zijn.

Een gemiddelde kost om een drone operationeel te houden wordt volgens [69] geschat op $0,021 per minuut, dit werd berekend met behulp van een studie van Deutsche bank [32] omgerekend is dit ongeveer 1,17 euro per uur. Voor wagens neemt hij een vaste (loon) kost van 20$ (18,49 euro) per uur, en voor kost voor een wagen $0,083 per min, wat omrekent naar ongeveer 4,60 euro per uur.

Met de loon kost erbij is dit een verschil van maar liefst 21,92 euro per uur, ofwel bijna 20 keer zo duur. Door dit enorme verschil zal een fitness functie die puur de kost minimaliseert bijna uitsluitend alles door drones laten doen. Hierdoor is enkel naar logistieke kost minimaliseren ook geen nuttige metriek, dan zouden we enkel drones gebruikt worden. Dit doet velen ook geloven dat de toekomst van goederentransport allemaal geautomatiseerd zal worden. In de studie van Deutsche bank [32] werd gesuggereerd dat Amazon tot wel 80% van zijn logistieke kosten kon besparen wanneer het volledig overschakelt naar drone delivery.

2.5.6 Fairness in DVRPD

Doordat we efficiënte routes verkiezen (bij dynamische VRPs) boven routes die garanderen dat we alle locaties zullen bezoeken (bij statische VRPs), komen we soms in de problemen met fairness bij het bedienen van klanten wat praktisch nadelen kan hebben. Zo zullen bij het efficiënt toewijzen van drones bijvoorbeeld telkens de dichtste locaties eerst behandeld worden. Hierdoor maximaliseren we de leveringen/uur, maar hierdoor zal het gebeuren dat mensen die dicht bij de depot wonen zeer snel behandeld zullen worden door drones. Maar mensen die in minder druk bevolkte delen ver van de depot wonen kunnen telkens pas laat bezorgd worden. En tijdens drukke perioden zou het zelfs kunnen gebeuren dat deze mensen zeer lang niet bezocht zullen worden wanneer er geen incentive komt om zo ver te gaan. Wanneer het dus wel belangrijk is dat het systeem fair is hebben we dus een extra incentive nodig voor deze verre plaatsen. Gelukkig hebben de systemen die we gebruiken namelijk job scheduling problems voor drones en team orienteering problem/ profitable tour problem beide systemen met profits per locatie. We kunnen dus door onze fitness functie aan te passen een fair systeem maken waarbij we in plaats van leveringen/uur proberen maximaliseren de profit/uur proberen maximaliseren. Dit zal uiteindelijk wel een afweging zijn aangezien onze leveringen/uur en dus efficiëntie van de operaties een hit zal krijgen door het opleggen van fairness.

Allereerst hebben we een vorm nodig van toekennen van profits, de meest voor de hand liggende manier is een functie die de profits voor een locatie verhoogt naargelang de tijd. Een andere vorm is om profits ook toe te kennen naar op basis van de afstand tot de depot. Een combinatie van de twee methodes is zeker ook een mogelijkheid. Een algemene formule voor de profits is als volgt:

$$P_i = 1 + (currTime - start_i) \cdot time_{const} + dist(L_0, L_i) \cdot dist_{const}$$

Hierbij zijn $time_{const}$ en $dist_{const}$ de tijd- en afstand-constante, deze vallen zelf in te vullen naargelang hoe fair je het systeem wil krijgen op basis van afstand en tijd.

Als we kijken naar de fitness functies dan kan de leveringen/uur fitness functie als volgt uitgebreid worden:

$$fitness_r = \frac{\sum_{l \epsilon r} P_l}{t_{min} + t_r}$$

En bij het algoritme voor het bepalen van de drones fitness vervangen we aantal pakjes, door de profits en sorteren we niet op afstand naar depot, maar op profit/uur en dus passen we ook onze toekenning strategie van drones aan van kortste afstand eerst naar hoogste profit/uur eerst. Daarna tellen we voor de totale fitness de profit/uur op voor de verschillende routes en drones.

Hoofdstuk 3

Gerelateerde literatuur

Veel van de gerelateerde literatuur waarop deze paper steunt en op verder bouwt, is ondertussen al eens ter sprake gekomen. In dit hoofdstuk proberen we een overzicht te geven van het onderzoek dat reeds gebeurd is. Door het actief onderzoek in dit vakgebied en de nieuwe aard van het probleem is er echter beperkt onderzoek in ons specifieke deelprobleem. Eerst bekijken we de de fundamanten waaruit deze tak van VRP met drones gekomen is en hierna kijken we welk onderzoek gebeurd is in het gebied van VRPD, DVRPD, PDSTSP, FSTSP, PDSVRP en andere varianten van vehicle routing problems met drones. We geven uiteindelijk ook een overzicht van alle literatuur en tonen aan welke bijdrage onze paper levert, en op welk onderzoek onze probleemstelling gebaseerd is. Deze literatuurstudie is voor een groot deel geïnspireerd door deze van Salue et al. [58] en Nguyen et al. [50].

Door de toenemende interesse in drone delivery door bedrijven en de paper van Murray et al [48] is drone delivery een hot-topic geworden binnen de TSP/VRP onderzoekswereld. Er zijn echter verschillende wegen en perspectieven gekomen in het onderzoek over het incorporeren van drones in het lever proces. Hierdoor zijn verschillende takken en deelproblemen ontstaan voor drone delivery. Allereerst hebben we twee grote varianten die we reeds behandeld hebben en die voor het eerst besproken werden in de paper van Murray en Chu [48], namelijk flying side-kick TSP (FSTSP) en parallel drone scheduling TSP (PDSTSP). Deze varianten van het probleem zijn adaptaties gemaakt binnen het onderzoeksgebied van VRPs en TSPs die het mogelijk maken ook drones te gebruiken bij de leveringen. Een andere variant van het drone-delivery probleem is er bij drone-only lever systemen, papers die dit onderzoeken zijn deze van Dorling et al [19], Sundar et al. [39]. Deze beschouwen het probleem waarbij er enkel een vloot van drones is en deze alle locaties moeten bezoeken vanuit de depot. Hierbij beschouwen ze echter meestal het probleem waarbij een drone meerdere pakketten kan transporteren, maar ook een limiet op gewicht, batterij en range heeft. Anders zou zoals eerder besproken dit probleem te reduceren vallen naar een parallel identical machine scheduling probleem.

Wanneer we echter een samenwerking tussen wagens en drones voorstellen is FSTSP de variant die het meest onderzocht is tot nu toe, en waar het onderzoek ook reeds het verste staat. Ook hier zijn onderlinge varianten tussen, de meeste werken met één wagen en één drone deze staat ook soms bekend onder de noemer TSP with drone (TSP-D) en wordt besproken en volgende papers: Agatz et al. [2], Ponza et al. [54], Daknama et al.[13], Marinelli et al. [43], Ha et al. [27], Yurek et al. [74], De Freitas et al. [15]. In deze papers worden heuristieken en meta-heuristieken voorgesteld voor het oplossen van HFTSP en hierdoor op een slim en efficiënte manier de oplossingsruimte te doorzoeken. Schermer et al. [60] bracht een exact banch-and-cut algoritme uit om het probleem op te lossen. Andere varianten nemen ook meer zaken in rekeningen zoals het leveren van meerdere pakketten per drone of een energy drain functie voor het gewicht van een drone zoals Cheng et al. [10] en Poikonen et al. [53].

Poikonen et al. [52] bracht later ook een andere interessante variant uit van TSP met drones. Een model met een mothership dat een aantal drones vervoert als een soort gigantische (vliegend) voertuig die n drones vervoert maar zelf geen pakjes levert.

Een andere variant van FSTSP is deze waarbij we een wagen hebben met meerdere drones per wagen. Hier kunnen meerdere drones tegelijk op een wagen zitten en van hieruit vertrekken, maar doet de wagen zelf ook leveringen. Over deze variant is onderzoek gedaan door onder andere Chang et al. [8] en Ferrandez et al. [22]. Zij stellen een cluster-based heuristiek voor om dit probleem op te lossen. Murray et al. [49] stellen het probleem met meerdere flying side-kicks voor als mFSTSP en stellen ook een MILP formulation voor, het oplossen van het probleem met MILP was echter zeer inefficiënt waardoor ze ook een 3-staps heuristiek voorstellen.

Sacramento et al. [57] en Wang et al. [68] generaliseerde hierna FSTSP voor m wagens in plaats dan één. Hierdoor hebben we dus routes voor m wagens met één drone side-kick. Ze noemden dit probleem VRP with drones (VRPD). Later werd door Pugliese et al. [18] hier een extensie op gemaakt die ook time-windows voor levering in acht neemt en dus een variant op vehicle routing problem with time windows maar dan met drones. Het geval waarbij we m wagens hebben en elke wagen heeft meerdere drones als side kick werd onderzocht door Kitjacharoenchai et al. [34]. Hierbij viel de contraint dat een drone altijd moest terug keren naar dezelfde delivery wagen ook weg, en kunnen drones dus bij een wagen vertrekken en een andere toekomen.

parallel drone schudling Traveling salesman problem (PDSTSP) krijgt in de literatuur echter een pak minder aandacht, ondanks de praktische mogelijkheden die er zijn voor het uitvoeren ervan. Vaak wordt het probleem dan ook opgelost door een opsplitsing te maken voor wagen routes en drone routes. Hierdoor bestaat het probleem uit twee deelproblemen die dienen opgelost te worden het TSP voor de wagen en het identical parallel machine scheduling problem (IPS) voor de drones. Hierdoor zijn er veel papers die het probleem reduceren naar een beslissingsprobleem waarbij er enkel een opdeling gemaakt dient te worden welke locaties laten we best door een wagen doen en welke best door een drone. Na het maken van deze opsplitsing worden de problemen onafhankelijk van elkaar opgelost met een gekend standaard TSP/VRP algoritme en IPS algoritme naar keuze. Deze strategie wordt toegepast door Murray et al. [48] en Ulmer et al. [62]. Murray et. al hebben hebben echter ook een MILP formulation gemaakt die het probleem exact kan oplossen, maar deze heeft veel tijd nodig om zelfs de kleinste instanties exact op te lossen waardoor problemen van praktische grootte er niet mee opgelost kunnen worden. Dell'amico et al. [17] stelde enkele matheuristics voor bij het oplossen van PDSTSP. En Kim et al. [33] maakten een variant van PDSTSP waarbij drones ook ingezet kunnen worden van andere drone deploy punten die dus kunnen verschillen van de depot, een soort multi-depot variant maar voor TSP. Hierna brachten Ham et al. [28] een multi-depot variant uit voor PDSVRP die ondersteuning had voor m wagens en n drones, deze had ook de mogelijkheid voor het werken met een drop & pick-up systeem en ondersteunt ook m-visit waarbij een drone dus meerdere plaatsen kan bezoeken. Bij drop & pick-up kan een drone na het leveren van een pakket een nieuw pakket ophalen bij een andere klant locatie dat geleverd dient te worden, de oplossing wordt gegeven met behulp van constraint programming. Wang et al. [67] bracht de ideeën van FSTSP en PDSTSP samen in een systeem waar drones zowel samen in tandem met een wagen als independent vanaf een depot ingezet konden worden. Dit systeem is ook mogelijk met m wagens en n drones, elke wagen kan hierbij maximaal één drone als sidekick hebben. Hier werd duidelijk dat het combineren van de 2 systemen een extra verbetering kan geven en het ene systeem niet gedomineerd wordt door het andere bij het optimaliseren van de performantie. Ulmer et al. [62] stelde hierna een dynamische variant voor die in een systeen van same-day delivery kan werken. Deze werd same-day delivery routing problem with heterogeneous fleets of drones and vehicles (SDDPHF) genoemd, en is de dynamische variant van PDSVRP die wij ook bespreken in hoofdstuk 5. Dit wordt gedaan om same-day delivery te faciliteren aangezien we hier een online algoritme nodig hebben. Hetgeen waar we hierbij optimaliseren

is het aantal pakketten geleverd met same-day delivery. Uit deze paper komt het initiële idee voor het schrijven van deze paper en de probleemstelling. Het algoritme dat voorgesteld wordt splitst het probleem op in twee deelproblemen die onafhankelijk van elkaar opgelost worden. Het algoritme is enkel een beslissingsalgoritme dat nieuwe locaties zal toewijzen aan drones of de wagens. Dit wordt gedaan met behulp van adaptive dynamic programming genaamd parametric policy function approximation (PFA).

Als we kijken naar het onderzoek in PDSVRP hebben we Nguyen et al. [50] en Conceição et al. [11] die een extensie hebben gemaakt van PDSTSP die ook meerdere routes van wagens toelaat en dus m wagens en n drones toelaten. Hetgeen waarin het echter verschilt in deze paper is dat de gegeven papers minimaliseren naar de totale transport kost. Die berekend wordt met een geschatte reis kost voor drones tegenover wagens. Er zijn echter enkele subtiele verschillen tussen de papers. Bij Nguyen zijn bijvoorbeeld constraints gelegd op de drone range, work duration en de wagen capaciteit en bij de ander niet. Beide stellen een MILP model voor bij het oplossen van het probleem. Bij Nguyen werd ook een heuristiek voorgesteld slack induction by string and sweep removals (SISSRs).
Saleu et al. [58] stelde PDSmTSP voor een variant op PDSVRP die een mTSP oplost in plaats van een VRP waarbij er dus geen maximum capaciteit is voor het aantal parcels dat een wagen kan vervoeren op een route. Hier wordt ook de makespan geminimaliseerd. Er wordt een MILP formulation voorgesteld en een hybride meta-heuristiek oplossingsmethode. Ray et al. [55] behandelt ook PDSVRP met m wagens en n drones en gebruikt hierbij een branch-and-price methode om dit op te lossen, er werden ook enkele heuristieken ontwikkeld om het probleem op te lossen.

In tabel 3.1 geven we een overzicht van alle besproken literatuur met de verschillende eigenschappen, hierbij stellen $\#V$ en $\#D$ respectievelijk het aantal wagens en het aantal drones voor. De papers waarmee we onze methode en oplossingen rechtstreeks kunnen vergelijken zijn de papers die als probleem PDSTSP hebben of werken met m wagens en n drones. Ham et al. [28] heeft echter een drop & pick-up systeem waardoor we hiermee niet kunnen vergelijken. Aangezien Conceição et al. en Nguyen et al. optimaliseren naar de kost kunnen we hier ook niet rechtstreeks mee vergelijken. De oplossingen waarmee we rechtstreeks hebben vergeleken hebben we aangeduid in het overzicht met een . We kunnen ook het aantal drones gelijkzetten aan één waardoor het probleem equivalent wordt met PDSTSP en we ook hiermee kunnen vergelijken. Maar aangezien ons algoritme hier niet op geoptimaliseerd is kan de kwaliteit van de oplossingen niet gegarandeerd worden. Dit aangezien we veel moves doen over verschillende routes en deze nu geen verschil zullen maken, maar andere moves hebben we nog steeds hun nut. In hoofdstuk 7 hebben we ook vergeleken met enkele van deze papers, tabel 7.10 hebben we ook vergeleken met de resultaten van Salue et al. [58] en in tabel 7.9 vergelijken we met de beste oplossingen van Salue et al. [46], Dell'Amico et al. [17] en Raj et al. [55].

3.1 Oplossing methoden voor VRP

Door de dynamische aard van het DVRPD probleem, en de grote omvang van de realistische scenario's gaan we voor een oplossingsmethode moeten zoeken met behulp van meta-heuristieken die ons op elk moment een goede oplossing kan geven. De keuze voor het vinden van een geschikte oplossingsmethode is niet makkelijk in de zin dat al bijna alles wel eens uitgeprobeerd is op VRP's en onze contraints ook veel nieuwe aspecten toevoegt aan het probleem. Belangrijk aan de oplossingsmethode is dat we snel een goede oplossing hebben, maar ook makkelijk kunnen blijven verder zoeken naar betere en nieuwe oplossingen zonder te blijven hangen in lokale minima. Zeker wanneer we onze methode willen uitbreiden naar dynamische variant is dit laatste zeer belangrijk. Zo kunnen we eerst in een zeer goede oplossing zitten, maar daarna door de nieuwe locaties of het wegvallen van een route kan er plots een veel betere nieuwe oplossing komen. Daarom moet de oplossingsruimte steeds grootgehouden worden, en zijn dingen als random restarts ook van groot belang om nieuwe betere oplossingen te vinden.

Tabel 3.1: Overzicht van gerelateerde literatuur

Reference	Problem	#D	#V	Objective function	Solution method
Murray et al. (2015) [48]	FSTSP	1	1	maksepan	MILP, greedy heuristic
Agatz et al. (2018) [2]	FSTSP	1	1	makespan	route-first, cluster-second heuristics based on LS and DP
Ponza (2016) [54]	FSTSP	1	1	makespan	simulated annealing
de Freitas et al. (2020) [15]	FSTSP	1	1	makespan	metaheuristic (HGVNS)
Ha et al. (2018) [27]	FSTSP	1	1	operational costs	MILP, metaheuristic (GRASP)
Murray et al. (2020) [49]	FSTSP	n	1	makespan	MILP, three-phased heuristic
Poikonen et al. (2020b) [53]	FSTSP	n	1	makespan	three-phased heuristic
Sacramento et al. (2019) [57]	FSTSP	n	m	operational costs	metaheuristic (ALNS)
Wang et al. (2019) [69]	PDSTSP/ FSTSP	n	m	makespan	three-step heuristic
Cheng et al. (2020) [10]	DDP	n	0	operational costs	branch-and-cut
Poikonen et al. (2020a) [52]	CVP-D	1	1	makespan	branch-and-bound, greedy heuristics
Mathew et al. (2015) [44]	CVP-D	0	1	total distance	decomposition, heuristic
Murray et at. (2015) [48]	PDSTSP	n	1	makespan	MILP, greedy heuristic
M. Saleu et al. (2018) [46]*	PDSTSP	n	1	makespan	iterative two-step heuristic
Dell'Amico et al. (2020) [17]*	PDSTSP	n	1	makespan	simplified MILP, matheuristic
Kim et al. (2018) [33]	PDSTSP	n	1	makespan	MILP, decomposition
Schermer et al. (2018) [59]	PDSTSP	n	1	makespan	MILP
Ham (2018) [28]	PDSTSP	n	m	makespan	constraint programming
Conceição (2019) [11]	PDSTSP	n	m	operational costs	MILP
Nguyen (2022) [50]	PDSTSP	n	m	operational costs	MILP, SISSrs
M. Saleu et al. (2022) [58]*	PDSmTSP	n	m	makespan	iterative two-step heuristic
Ulmer et al. (2018) [62]*	SDDPHF	n	m	#SDD nodes	Adaptive dynamic programming, PFA
Raj et al. (2021) [55]*	PDSVRP	n	m	makespan	MILP, Branch-and-price
This paper	PDSVRP	n	m	makespan	Genetic algorithm (HGS)

Wat volgt is een oplossingsmethode die succesvol gebleken is op large-scale VRPs, deze behoort tot de huidige state-of-the-art en behoort tot de beste in zijn categorie. Dit werd bepaald met behulp van de CVRPLIB [71], een platform dat benchmark data voor cVRP aanbiedt en de beste oplossingen voor de verschillende instanties bijhoudt. Onderzoekers kunnen hun oplossingen insturen, samen met de paper of uitleg van hoe ze deze bekomen zijn. Op deze manier kunnen we de huidige state-of-the-art makkelijk bijhouden, waar de volgende oplossingsmethoden toe behoren.

3.2 Hybrid genetic search (HGS)

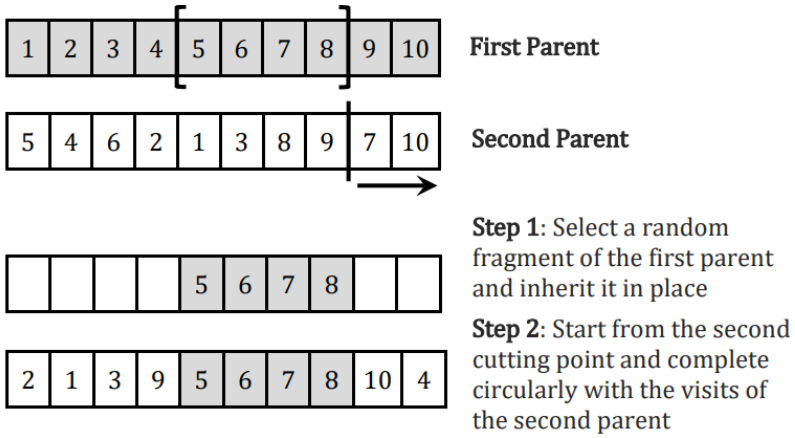
Hybrid genetic search with adaptive diversity control werd door Vidal et al. geïntroduceerd in zijn paper "A hybrid genetic algorithm for multi-Depot and periodic vehicle routing problems"[66]. Deze techniek gebruikt een hybride methode waarbij een populatie bijgehouden wordt van feasible en infeasible oplossingen waartussen cross-overs gedaan wordt en verder gezocht met localsearch. Initieel was het doel van de paper een state-of-the-art methode uit te werken voor drie varianten van het VRP, namelijk multi-depot VRP (MDVRP), periodic VRP (PVRP) en multi-depot periodic VRP (MDPVRP). Bij multi-depot VRP werken we met een aantal locaties, maar ook een aantal depots. Elke depot heeft een vloot van voertuigen en er moet dus zoals bij het VRP een efficiënte toewijzing van routes voor de koeriers zijn, maar hier bovenop moet bij multi-depot, ook een toewijzing gedaan worden van welke depot welke locatie zal afhandelen. Bij PVRP wordt een generalisatie gemaakt waarbij we de periode van de VRP zolang of zo kort maken als we willen, meeste VRPs werken over een periode van één dag, maar bij periodic VRP kunnen we werken over n dagen, elke locatie heeft een daily demand, maar deze kan ook na n dagen ingelost worden. We moeten dan een indeling vinden waarvoor de daily demand altijd vervult is. We proberen te optimaliseren naar de som van de reistijd van de koeriers. Het is de bedoeling dat we een VRP uitkomen waarmee we periodisch over een periode van n dagen de dagelijkse vraag van alle locaties kunnen invullen.

De oplossingsmethode bleek zeer succesvol te zijn en ook bruikbaar voor het standaard cVRP, Vidal et al. bracht ook later een paper uit waarin hij het algoritme had geoptimaliseerd naar het cVRP en bracht hierbij ook een open-source implementatie uit. De grote stappen van HGSADC staan beschreven in algoritme 2. Bij HGSADC zullen we twee populaties opbouwen één met feasible en een andere met infeasible oplossingen. Deze worden gemaakt door een cross-over te doen van oplossingen gevolgd door een localsearch van het resultaat. Hierna wordt bepaald of dit een feasible of infeasible oplossing is en aan de bijhorende populatie gevoegd. Hierna worden de survivors geselecteerd. De manier waarop we oplossing evalueren gebeurt volgens een biased fitness functie, deze neemt de doelfunctie in acht samen met een diversiteitscontributie die een oplossing levert aan een populatie. Op deze manier behouden we een populatie die tegelijk goede oplossingen bevat, maar ook een ruime zoekruimte behoudt. Anders gebeurt het snel dat onze populatie enkel zeer lokale variaties van de best known solution zal bevatten en we in locale minima blijven hangen. De diversiteit contributie definiëren we als de gemiddelde afstand tot de n dichtstbijzijnde buren. De afstand tussen twee oplossing wordt berekend door het aantal successors en predecessors in de route die overeenkomen.

$$D(I_1, I_2) = \frac{1}{2n} \sum_{i=1}^{n} (pred(l_i \in I_1) \neq pred(l_i \in I_2)) + (succ(l_i \in I_1) \neq succ(l_i \in I_2))$$

De afstand tussen twee oplossingen wordt dus geïncrementeerd elke keer dat een locatie l_i dezelfde successor of predecessor heeft in oplossing I_1 en I_2. Uiteindelijk wordt genormaliseerd door te delen door $2n$. De diversiteitscontributie wordt gegeven door de neighbourhood N_{close} te bepalen van de oplossing met oplossingen met de kortste afstand en hier het gemiddelde van te bepalen.

$$dc(I) = 1/|N_{close}| \sum_{I_1 \in N_{close}} D(I, I_1)$$



Figuur 3.1: OX Cross over illustratie uit Vidal et al. [65]

$$BF(I) = fitness(I) + (1 - \frac{\#elite}{\#population})dc(I)$$

Algorithm 2 HGS-cVRP Vidal et al. [66]

```
Initialize population with random solutions improved by localsearch;
while number of iterations without improvement < It_NI and time < T_max do
  Select parent solutions P_1 and P_2;
  Apply the crossover operator on P_1 and P_2 to generate an offspring C;
  Educate offspring C by localsearch;
  Insert C into respective subpopulation;
  if C is infeasible then
    With 50% probability, repair C (localsearch) and insert it into respective subpopulation;
  end if
  if maximum subpopulation size reached then
    Select survivors;
  end if
  Adjust penalty coefficients for infeasibility;
end while
return best feasible solution;
```

De cross-over wordt gedefinieerd volgens het idee van cross-over with insertions, en gaat als volgt. Allereerst bepalen we twee parents op welke we de cross-over willen toepassen. Dit doen we via een binarytournament waarbij we twee oplossingen random kiezen en degenen met de beste biased fitness gebruiken voor de cross-over. Hierna bepalen we random plaatsen waar we de cross-over willen starten en stoppen in de eerste parent. Deze delen van de cross-over kopiëren we rechtstreeks over in ons resultaat. We houden bij in een set welke locaties we al bezocht hebben. We kopiëren van de tweede parent de locaties over die we nog niet bezocht hebben in het cross-over gedeelte van de eerste parent en dit in dezelfde volgorde, we skippen hierbij nodes die we al bezocht hebben. Wanneer we dan de start locatie van de sequentie van de eerste parent tegenkomen stoppen we, en hier komt de sequentie van de eerste parent, daarna gaan we terug verder. Deze vorm van cross-over hanteren we ook in het genetisch algoritme dat wij gemaakt hebben, en wordt uitvoerig besproken en sectie 4.7 en geïlustreerd in figuur 3.1.

Na een cross-over doen we een localsearch op het bekomen resultaat. Vaak hebben deze cross-overs

namelijk veel potentieel, maar zitten deze nog niet in een lokaal optimum en zouden ze daarom niet overleven in een sterke populatie mochten ze niet in een lokaal optimum zitten. We gaan dus na een cross-over het lokale optimum proberen vinden. De localsearch bestaat uit drie fases: route improvement gevolgd door pattern improvement waarna opnieuw route improvement. Bij route improvement itereren we random over de vertices u en de neighbors van de vertices v waarbij de vertices v de vertices zijn die het dichtste bij v liggen. Hierbij doen we random volgende moves:

- (M1) Als u een locatie bezoekt, verwijder u en plaats het na v;
- (M2) Als u en x locaties bezoeken, verwijder ze dan en plaats u en x na v;
- (M3) Als u en x locaties bezoeken, verwijder ze en plaats x en u na v;
- (M4) Als u en v locaties bezoeken, swap u en v;
- (M5) Als u, x en v locaties bezoeken, swap u en x met v;
- (M6) Als u, x, v en y locaties bezoeken, swap u en x met v en y;
- (M7) Als $r(u) = r(v)$, vervang (u, x) en (v, y) door (u, v) en (x, y);
- (M8) Als $r(u) \neq r(v)$, vervang (u, x) en (v, y) door (u, v) en (x, y);
- (M9) Als $r(u) \neq r(v)$, vervang (u, x) en (v, y) door (u, y) en (x, v);

Een andere move die wordt besproken in de cVRP implementatie is de SWAP* move die gaat in tegenstelling zoals bij een swap move (M4, M5) twee nodes swappen, maar zonder in-place insertion. Concreet als vertex v in route r SWAP* ondergaat met v' in route r', wordt v ingebracht op de beste positie van r' en v' ingebracht op de beste positie van r. We kunnen SWAP* in kwadratische tijd uitvoeren, en met extra optimalisaties zoals enkel kijken naar plaatsen waar de poolcirkels van de routes overlappen kunnen we dit zelfs reduceren naar sub-kwadratische tijd.

Bij pattern-improvent proberen we de patronen te verbeteren, wat specifiek is voor PVRP, en de verschillende patronen weergeeft waarop we een customer kunnen bezoeken over de gegeven periode met een gegeven service frequentie van de klant. Omdat dit niet belangrijk is voor cVRP of ons probleem gaan we hier niet verder op in.

De manier waarop we de routes hier voorstellen en construeren is als één grote route die bestaat uit alle routes na elkaar uit te voeren. Dit omdat we op deze manier makkelijk de cross-over kunnen definiëren over meerdere routes, we kunnen erna ook met behulp van het split algoritme uit Vidal et al. [64] deze grote route ideaal opdelen in meerdere routes, deze reduceert het probleem naar een kortste pad probleem op een acyclische graaf. Hierdoor kan dit probleem opgelost worden in polynomiale tijd $O(m\, n^2)$.

Hoofdstuk 4

Hybrid genetic search voor drones (HGSD) voor PDSVRP

Als oplossingsmethode voor PDSVRP en DVRPD hebben we gekozen voor een genetisch algoritme gebaseerd op HGSADC uit Vidal et al. [66] en Euchi et al. [20], met uitgebreide population management, diversification management en nieuwe mutaties. Vele concepten uit de paper van Vidal et al. [66] worden onderzocht en vergeleken met alternatieven. Door vereenvoudiging van het VRP zijn er echter nieuwe paden die opengaan op vlak van mutaties en localsearch. Zo worden enkele nieuwe mutaties mogelijk, en kunnen we nieuwe zoek strategieën implementeren.

4.1 Genetic search

Bij onze zoektocht naar een goede benaderde oplossing gebruiken we een genetische manier van werken. Deze bouwt een populatie op van mogelijke oplossingen waaraan we blijven verder werken. Zoals in een klassiek genetisch algoritme passen we elke ronde elitisme toe op de populatie waardoor we enkel elke generatie de beste $s\%$ mutaties overhouden, dit noemen we de survivors. De andere $1 - s\%$ wordt verwijderd. Hierna werken we verder op de survivors en voegen de $1 - s\%$ die verwijderd is verder aan met nieuwe mutaties van de survivors, zo hebben we een vorm van evolutie en zoeken we enkel verder op onze beste mutaties. We doen dit tot een bepaald eind criterium behaald wordt. Dit kan een time constraint zijn zoals tijd, maar ook een bepaald aantal iteraties zonder verbetering te vinden op de beste oplossing. Tijdens de zoektocht houden we onze zoekruimte graag vrij breed, en proberen we aan de hand van random mutaties en cross-overs goede localsearch toe te passen en in lokale minima terecht te komen. Door een diverse populatie bij te houden en hierop cross-overs en andere (grote) mutaties te definiëren hebben we een mechanisme om uit lokale minima te raken en in het globale minimum terecht te komen. Er zijn echter wel enkele zaken van essentieel belang om deze methode te doen slagen.

Allereerst hebben we een goede population management nodig die tegelijk ons snel goede resultaten kan geven, maar met meer werk ook veel dichter bij optimale oplossingen kan geraken, zonder vast te blijven zitten in lokale minima. Het is hierbij dus belangrijk dat we niet enkel proberen verder zoeken vanaf onze beste oplossing, maar ook verder zoeken aan andere oplossingen in onze populatie. Ten tweede moeten we een veel voorkomend probleem bij deze genetische algoritmen zien te voorkomen, namelijk dat de diversiteit van de populatie groot blijft en we niet uitkomen met populaties die allen boven hetzelfde minima hangen. Dit is van essentieel belang om niet in lokale minima te blijven hangen, maar zeker ook om nuttige cross-over operaties uit te voeren. Een cross-over tussen 2 bijna identieke oplossingen is meestal ook niet nuttig. Een laatste, maar belangrijke voorwaarde voor het slagen is de performantie en snelheid van de mutaties en cross-over. We willen namelijk een manier waarop we snel

convergeren naar lokale minima, maar ook zeer veel mutaties uitproberen. Daarom moeten deze snel kunnen gebeuren, maar liefst ook slim, zodat we vooral mutaties kiezen die het meeste kans hebben verbeteringen teweeg te brengen. Uiteindelijk is het ook belangrijk om genoeg randomness te hebben in de mutaties en cross-over. Wanneer we bijvoorbeeld telkens de best mogelijke lokale keuzes maken, komen we al snel in dezelfde lokale minima terecht. Soms kan een lokale suboptimale keuze maken zorgen voor een uiteindelijke verbetering, en dit zorgt ook dat we meer delen van de oplossingsruimte zullen doorzoeken.

Dit kunnen we verwezenlijken door slimme heuristieken toe te voegen bij het kiezen van onze mutaties. Maar ook hier is vaak de afweging tussen de kost om de heuristiek uit te voeren ten opzichte van een random mutatie. Als random bijvoorbeeld een grootte orde sneller is kan het soms beter zijn gewoon random mutaties te pakken van een bepaalde soort. Ook komen te veel heuristieken soms de oplossing niet ten goede omdat onze zoekruimte erdoor te veel ingeperkt wordt. Er dient dus een afweging gemaakt te worden tussen efficiëntie/snoeivermogen van de mutaties en randomness om een grote zoekruimte te kunnen blijven doorzoeken. Want de sterkte van een genetisch algoritme zit hem net in generaties van random mutaties blijven verbeteren. De pseudocode van een generisch genetic search algoritme wordt gegeven in algoritme 3.

Algorithm 3 General genetic search

```
population ← [];
for i between 0..populationSize do
    solution ← random initial solution;
    population.add(solution);
end for
iteration = 0;
while not stopcondition do
    sort population by fitness;
    for i between #survivors..populationSize do
        r ← random number between 0 and 100;
        if r < 10 then
            donor ← random solution in survivors;
            acceptor ← random solution in survivors;
            population[i] ← crossOver(donor, acceptor);
        else
            parent ← random solution in survivors;
            population[i] ← mutate(parent);
        end if
    end for
end while
sort population by fitness;
return population[0];
```

4.2 Initiële oplossing

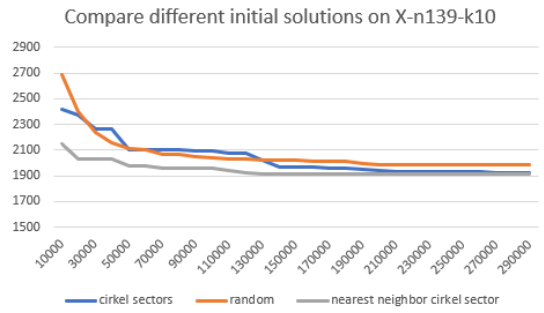
Een belangrijk onderdeel voor elke localsearch methode is de initiële oplossing waaruit we starten. Zeker wanneer we een methode hebben met random restarts zoals bij ons is het belangrijk dat wanneer we een nieuwe, initiële oplossing genereren deze aan enkele criteria voldoet. Het doel van een initiële oplossing is om het algoritme op een degelijke manier te kunnen starten waardoor we een deel van het werk van het localsearch algoritme al kunnen doen. Wanneer we namelijk van een compleet random

start beginnen zal het al een tijd duren bij grote instanties voor we een degelijke oplossing hebben waar alle locaties van de routes relatief dicht bij elkaar liggen. Wanneer we al een oplossing kunnen geven in een snelle tijd waar deze initiële zaken reeds gebeurt zijn kunnen we hiermee veel tijd besparen en onze tijd aan nuttigere dingen besteden en echte lokale minima zoeken.
Een andere belangrijke eigenschap zeker bij een genetische algoritme (GA) en zeker ook wanneer we veel heuristieken gebruiken is dat er genoeg randomness in de initiële oplossingen zit. Wanneer we bijvoorbeeld telkens van een deterministische gretige oplossing zouden starten, zal onze initiële populatie al totaal geen diversiteit hebben en uit dezelfde oplossingen bestaan wat geen goede start staat is. En wanneer we dit doen in combinatie met veel heuristieken is de kans dat we telkens in hetzelfde minima terecht komen zeer groot.
Een andere belangrijke eigenschap is dat we genoeg ruimte voor verbetering geven, dit kan wat contradictorisch lijken dat we en een goede oplossing moeten maken, maar deze ook niet te goed mag zijn. De reden hiertoe is vrij simpel. Als we starten van een te goede oplossing wordt de zoekruimte veel beperkter om verbeteringen te vinden waardoor we hoogstwaarschijnlijk enkel in het minima boven de initiële oplossing uitkomen. Wanneer we wat lager starten kunnen we vaak nog meerdere kanten op en in verschillende minima terecht komen.

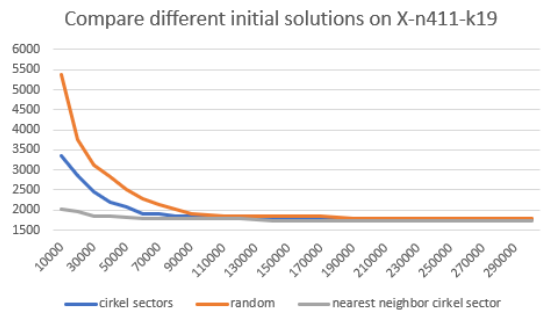
We hebben dus een oplossing nodig die zoveel mogelijk initieel werk wegneemt van het GA, die random genoeg is, maar ons genoeg ruimte geeft om te kunnen exploreren en in verschillende minima terecht te komen. We hebben ook verschillende manieren getest en met elkaar vergeleken, de belangrijkste heuristiek die we gebruikt hebben bij het maken van de oplossingen is dat dichte opvolgende locaties en clusters van locaties in dezelfde buurt in dezelfde route steken vaak voor goede routes zorgt. Dit zijn de initiële oplossingen die we geprobeerd hebben:

- **Random oplossing:** Start van een compleet random oplossing waarbij we alle nodes random met gelijke aantallen verdeeld hebben over de wagens en drones.
- **Route per cirkelsegment:** Hierbij hebben we de locaties opgedeeld in groepen van gelijke grootte volgens cirkel segment. Concreet sorteren we de locaties op basis van de hoek die ze maken met de depot. En verdelen deze gesorteerde lijst gelijk over de verschillende routes. Om randomness te introduceren starten we altijd te verdelen vanuit een random positie in de lijst.
- **Nearest neighbor route:** Hierbij maken we onze route door iteratief telkens de dichtste vrije locatie te nemen bij de laatste locatie van onze route en deze achteraan de route toe te voegen tot alle locaties verdeeld zijn. Hierbij kunnen we op 2 manieren toewijzen ofwel laten we de routes 1 voor 1 hun volgende locatie in de route nemen ofwel werken we route per route en vullen we telkens een route volledig en gaan naar de volgende route. Om hier randomness te introduceren kunnen we een kans geven tussen de eerste n dichtste.
- **Nearest neighbor in cirkelsegment:** Hierbij delen we opnieuw de locaties op volgens cirkel segment, maar voor de route in het cirkel segment zelf gebruiken we de nearest neighbor aanpak.

We merken vooral grote verschillen bij de oplossing kwaliteit bij grote routes. Wanneer we namelijk bij grote routes ergens slechte keuzes maken in het begin is het soms lastig deze terug goed te krijgen, dus als we reeds van degelijke start posities starten kunnen we makkelijker in goede oplossingen terecht komen en verliezen we geen kostbare tijd. Een vergelijking kan gevonden worden in afbeelding 4.1. We zien duidelijk dat random na 10k iteraties nog steeds een pak boven de rest zit, daar dit meestal uiteindelijk wel goed komt. Maar als we kijken naar de grafiek van X-n411-k19 zien we dat de nearest neighbor in cirkelsectors al snel aan een bijna optimaal level zit, uiteindelijk haalt deze gemiddeld gezien ook een lagere score na 300k iteraties. Als we het gemiddelde nemen over 10 uitvoeringen haalt nearest neighbor in cirkelsectors hier 1764,23, random 1796,65 en cirkelsegment 1795,89. Dus onze goede start leidt op termijn wel nog steeds voor een betere oplossing. Bij kleinere instanties zoals X-n139-k10 zien we dat na 100k iteratie we niet zo ver meer van elkaar zitten, maar dat de goede starts uiteindelijk wel meer in goede oplossingen terecht komen. Als we hier het gemiddelde nemen over 10 runs komen

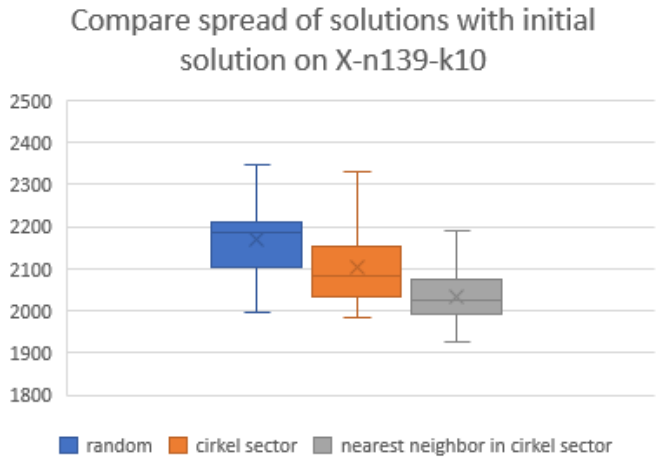


(a) X-n139-k10



(b) X-n411-k19

Figuur 4.1: Vergelijking van de beginoplossingen van random, cirkelsector en nearest neighbors in circlesector



Figuur 4.2: Vergelijking van oplossing spread voor random, cirkelsector en nearest neightbors in circlesector initiële oplossingen met 100K iteraties op 25 runs

we aan 1978,56 voor nearest neighbor in cirkelsectors, 1988,05 voor nearest neighbor en 2121,34 voor random. We zien als we kijken naar de spread van de makespan van verschillende oplossingen na 100k in figuur 4.2 dat we met nearest neighbor in cirkelsectors consistenter betere resultaten halen, maar bij cirkel sectoren kunnen we soms in slechte of net heel goede paden terecht komen, daarom is de uiteindelijke makespan hier soms een pak verschillend, maar kan deze nog beter worden zoals te zien in figuur 4.1 en voegt deze ook meer randomness toe, daarom gebruiken we meestal cirkel sectoren voor de tests. Enkel bij zeer grote instanties gebruiken we nearest neighbor in cirkelsectors omdat we hier vaak zeer traag convergeren en dus helpt nearest neighbor in cirkelsectors hier veel bij. Op vlak van performance is de tijd om de initiële start te vinden insignificant ten opzichte van het localsearch proces, dus de tijd die we in het maken van de initiële oplossing steken is het altijd waard.

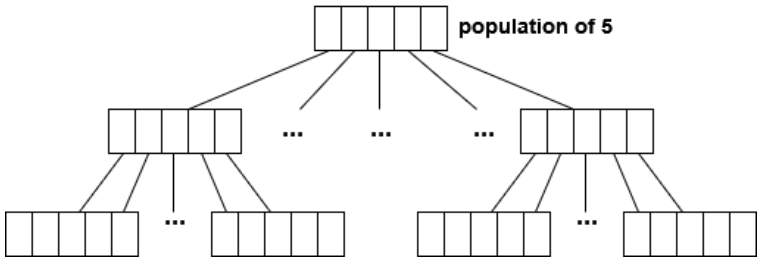
4.3 Population management

Het is van essentieel belang voor het slagen van het algoritme dat de populatie te allen tijde streeft naar diversiteit, maar dit mag niet te veel ten koste gaan van performantie. We initialiseren onze

populatie met een initieel gegenereerde oplossing, dit kan een random oplossing zijn, of een random oplossing bekomen met een bepaalde heuristiek. Hierna doen we telkens elitisme volgens een bepaalde biased fitness functie, de beste s procent halen dan de volgende generatie. Hier worden op random oplossingen van de populatie random mutaties en cross-overs uitgevoerd. De biased fitness functie neemt twee aspecten van het probleem in acht. Ten eerste de algemene fitness van een oplossing wat overeenkomt met de doelfunctie waarnaar we optimaliseren, en ten tweede met de diversiteit van de oplossing. Hoe we dit precies gedaan hebben wordt uitvoerig besproken in de volgende sectie over de diversification management.
Door een random start gebeurt het snel dat oplossingen die het snelste convergeren ondanks het in acht nemen van de diversiteit toch de populatie gaan overheersen. Dit omdat door de random start we zeer slechte initiële scores hebben die snel een veel betere fitness kunnen krijgen. Hierdoor overheerst de fitness component de diversificatie component en zullen we alsnog een populatie hebben die grotendeels uit dezelfde eerste beste voorvaderen stammen. Daarom is er nood aan extra maatregelen om de algemene diversiteit te bevorderen.

Een mogelijke oplossing voor dit probleem is het introduceren van random restarts: momenten tijdens de localsearch waarbij we terug van een random oplossingen starten en nieuwe oplossingen zoeken. Het is echter zeer gemakkelijk om deze nieuwe oplossingen samen te voegen met een andere oplossingen gevonden in het genetisch algoritme door de beste oplossingen gevonden in de random restart toe te voegen in de populatie van dit GA. We kunnen dus makkelijk oplossingen van verschillende random restarts samennemen en een nieuwe populatie vormen waarop we verder kunnen gaan met ons GA. De manier waarop we locaties van een random restart introduceren is ook vrij belangrijk, aangezien deze snel terug uit de populatie kunnen vallen als de fitness een pak minder is. Er dient dus opgelet te worden dat oplossingen die bij elkaar gevoegd worden van gelijkaardige kwaliteit zijn en voldoende divers. Op deze manier bekomen we de beste oplossingen met de meest diverse populatie.

4.3.1 Hiërarchisch population management



Figuur 4.3: Full hierarchy population management

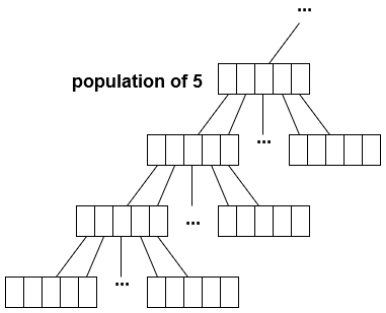
Hiervoor is er een methode ontwikkeld die in plaats van één populatie een hiërarchie van populaties zal bijhouden. Dit is een boomstructuur van populaties waarbij enkel de beste kinderen doorstromen naar de volgende populatie. Op deze manier maken we ook populaties die niet starten van volledig random oplossingen, maar wel oplossingen die al enkele generaties onafhankelijk van elkaar getraind zijn. Op deze manier kunnen we ook vaak meer halen uit cross-overs doordat sommige oplossingen bepaalde delen of routes al zeer efficiënt zijn en andere totaal niet. Diversiteit valt dan ook beter in te schatten omdat we kunnen zien of deze oplossingen van dezelfde tak komen en dus al samen in een populatie gezeten hebben. Bladeren van de hiërarchie worden gegenereerd met nieuwe random restarts. We kunnen verschillende contraints geven voor het overgaan naar het volgende level van de hiërarchie. De meest voor de hand liggende zijn een maximumaantal generaties per populatie of een tijd contraint. Oplossingen uit hogere levels van de hiërarchie worden gevuld met goede oplossingen van de kinderen.

Erna kan ook op deze nieuwe populatie het genetische algoritme toegepast worden. Zo kunnen we naar boven blijven propageren en op elk level betere oplossingen bekomen. Het uitvoeren van random restarts en zoeken in lage-orde populaties zijn wel kostelijk voor vaak slechts weinig verbetering in fitness. Er zijn enkele zaken die we zelf kunnen kiezen: de populatie grootte p, het aantal kinderen per node k, en het aantal generaties of levels in de hiërarchie g. Het aantal populaties in de hiërarchie is dan gelijk aan $\sum_{i=0}^{g-1} k^i$. Het aantal nodes in functie van het aantal generaties stijgt dus exponentieel. Maar het voordeel van deze techniek is echter wel dat de oplossingen die samengenomen worden in een nieuwe populatie altijd evenveel iteraties hebben trainen en gelijke kansen gekregen. Een nadeel is echter dat we moeilijk verder kunnen blijven zoeken na het bereiken van de top van de hiërarchie. We kunnen random starts blijven invoegen die minder generaties hebben kunnen draaien of we kunnen een extra generatie maken, maar dit eerste zal minder nog voor verbeteringen zorgen en een level toevoegen betekend dat we k nieuwe takken moeten toevoegen die evenveel werk hebben gedaan als de vorige dus vermeerdert ons werk telkens met een factor k elk level we toevoegen wat dus totaal niet schaalbaar is.

Deze methode heeft echter wel veel tijd nodig om tot een eindresultaat in een lokaal minimum terecht te komen. Daarom is het in sommige gevallen waar we ten alle tijden een goede oplossing moeten kunnen geven of beperkt zijn in tijd, beter eerst een goede oplossing te bekomen waar we geen verbeteringen meer op vinden. Hierna kunnen we de hiërarchie opbouwen.

Op deze manier kunnen we goede oplossingen makkelijker van elkaar laten leren, anders hebben we vaak het probleem dat we na een aantal iteraties met een oplossing bezig zijn en allemaal afstammelingen van deze oplossing met weinig diversiteit. Als we met deze populatie vol goede verschillende oplossingen starten kan vaak aan meerdere verschillende oplossingen gewerkt worden en leren de oplossingen goede stukken van elkaar. Zeker voor cross-over operaties is dit zeer belangrijk.

4.3.2 Ketting-hiërarchische population management



Figuur 4.4: Hierarchy chain population management

Wanneer we echter in een dynamische setting zitten of de mogelijkheid willen om te blijven zoeken naar een oplossing dan kunnen we afstappen van een volledige hiërarchie op te bouwen. We kunnen in plaats daarvan een hiërarchie bouwen waar we telkens voortbouwen op de beste populatie. We starten van een random populatie en laten deze een aantal iteraties uitvoeren. Hierna laten we de beste $s\%$ van deze generatie doorgaan naar het volgende level. Daarna doen we k random restarts en laten deze ook een aantal iteraties trainen, dan voegen we de beste oplossingen van de random restarts bij de initiële populatie toe en vullen zo de overige $1-s\%$ op met deze oplossingen. Op deze manier bekomen we een nieuw level dat voortbouwt op de vorige samen met nieuwe oplossingen van random restarts. Op deze manier bouwen we een soort ketting waar we makkelijk een extra level kunnen toevoegen. Dit herhalen we telkens, we laten steeds de populatie trainen en de beste $s\%$ doorgaan naar de volgende generatie en vullen de rest aan met random restarts. Hierdoor hebben we een hiërarchie waar een kind de volledige geschiedenis is van de hiërarchie en de andere kinderen random restarts. Op deze manier

zijn er slechts $(g-1)\cdot k+1$ nodes in de hiërarchie en kunnen we makkelijk blijven verder zoeken door k random restarts te doen en samen te voegen in een nieuwe populatie. Het voordeel hierbij is dat we veel tijd spenderen aan het verbeteren van de beste oplossingen en dus vrij snel goede oplossingen krijgen, ook is verder zoeken vrij eenvoudig en schaalt dit veel beter. Bij een volledige hiërarchie schaalt het verder zoeken exponentieel met een factor k, maar bij onze ketting-hiërarchische structuur schaalt dit lineair. Een nadeel is echter wel dat na een tijd de random restarts een veel slechtere fitness kunnen hebben dan de hoofd populatie en dus zelden voor verbeteringen zullen zorgen, een oplossing hiervoor is naargelang de fitness van de populatie groter wordt, we ook meer iteraties rekentijd geven aan de random restarts. Voor kleine en middelgrote oplossingen kunnen we vaak snel een lokaal minimum vinden bij een random restart en valt dit probleem weg. Daarom kunnen we als constraint voor het trainen van de hoofdpopulatie in de tests een maximumaantal iteraties zonder verbetering gebruiken, zodat we in een lokaal optimum terecht komen. We raken hier potentieel uit door cross-overs en dergelijke met de nieuwe oplossingen in de hoop dat we in globale minima terecht kunnen komen. Bij grote problemen is het echter soms niet mogelijk telkens tot een lokaal minima te blijven zoeken en zetten we vaak ook een maximumaantal iteraties hierop.

Doordat we onze oplossingsmethode schaalbaar en mogelijk voor dynamische problemen willen maken, was de keuze voor de techniek met de ketting-hiërarchie vrij logisch. Zeker in het dynamische geval is het onhaalbaar te blijven werken in een volledige hiërarchie. De ketting-hiërarchie voldoet hier wel aan alle eigenschappen die we nodig hebben.

4.4 Diversification management

Zoals eerder besproken is diversiteit behouden in een populatie zeer belangrijk om de zoekruimte groot genoeg te houden en niet vast te zitten in lokale minima. Als we hier geen rekening mee houden en enkel kijken naar fitness, komen we vaak terecht in een situatie waar we na een groot aantal generaties een populatie hebben waarin alle oplossingen in de populatie bestaan uit zeer kleine mutaties van de huidige beste oplossing. Hierdoor hebben deze oplossingen allemaal een zeer hoge fitness, maar de diversiteit is dan enorm laag. We hebben dus een manier nodig om de diversiteit in onze populatie te behouden waardoor we meerdere opties openhouden om uiteindelijk enkel met de beste opties verder te laten gaan.

4.4.1 Adaptive diversity control

Een mogelijke oplossing hiervoor wordt voorgesteld door Vidal et al. in zijn paper over hybrid genetic search with adaptive diversity control (HGSADC) [66]. Hier wordt een systeem van adaptive diversity control (ADC) voorgesteld waarbij een diversiteitscontributie ΔP bijgehouden wordt die de gemiddelde afstand voorstelt tussen een oplossing en zijn n dichtste buren. De diversificatie zelf wordt bepaald door een genormaliseerde Hamming distance $d_H(P_1, P_2)$ tussen de oplossingen P_1 en P_2. Deze geeft aan hoe gelijkaardig de oplossingen zijn aan de hand van hoeveel successors en predecessors er dezelfde zijn in de oplossing voor elke locatie.

$$d_H(P_1, P_2) = \frac{1}{2n}\sum_{i=1}^{n}(pred(L_i \in P_1) \neq pred(L_i \in P_2)) + (succ(L_i \in P_1) \neq succ(L_i \in P_2))$$

Hierbij is $pred(L_i \in P_1)$ de predecessor van customer i in oplossing P_1, en $succ(L_i \in P_1)$ de successor van customer i in oplossing P_1. De predecessor en succesor conditie geeft 1 terug wanneer deze true is en 0 wanneer deze false is. $d_H(P_1, P_2)$ is een waarde tussen 0 en 1 die 0 is wanneer alle successors en predecessors gelijk zijn en 1 als geen enkele gelijk zijn. De diversiteitscontributie wordt gegeven door de gemiddelde diversiteit tussen de dichtste buren N_{close} die het minst divers zijn van de oplossing P.

$$\Delta P = 1/|N_{close}| \sum_{P_1 \in N_{close}} D(P, P_1)$$

De biased fitness BF wordt gegeven door:

$$BF(P) = fitnessRank(P) + (1 - p_{elite}) \cdot \Delta P$$

Hierbij is $fitnessRank(P)$ de rank van de oplossing P krijgt na sorteren volgens de fitness functie en dus hetgeen waarnaar we willen optimaliseren en p_{elite} het percentage survivors in een generatie van het genetische algoritme. Met ADC hebben we een nieuwe biased fitness functie die ons een score kan geven die niet enkel de fitness in acht neemt, maar ook kijkt hoeveel deze oplossing bijdraagt aan de totale diversiteit van de populatie. Op deze manier gaan niet enkel de beste oplossingen door, maar ook goede oplossingen die diversiteit toevoegen aan onze populatie. Op deze manier houden we ruimte voor andere oplossingen om ook te verbeteren in de populatie zonder er direct uitgesmeten te worden door kleine mutaties van betere oplossingen. Dit is echter wel een afweging die gemaakt dient te worden, aangezien slechtere oplossingen in de populatie houden vaak ook voor veel onnuttig werk kan zorgen. Als deze slechtere oplossing niet verbeterd is er geen pay-off en was de moeite voor niks. Dit is wel een pay-off die we altijd zullen moeten maken om exploratie mogelijk te maken. Want wanneer alle oplossingen zeer dicht bij de beste liggen gaan alle resources naar de beste oplossing verbeteren en zullen we sneller verbetering krijgen. Diversificatie houdt dus de zoekruimte breder ten koste van efficiëntie, maar op termijn kan dit wel voor betere globale oplossingen zorgen.

4.4.2 Diversity control met Levenshtein distance

De genormaliseerde Hamming distance gebruiken als diversiteitsmetriek tussen twee oplossingen is een goede vereenvoudiging om de afstand te kennen tussen 2 oplossingen, maar een meer accurate metriek hiervoor is de Levenshtein distance zoals beschreven in Levenshtein et al. [38]. Deze wordt gebruik bij strings om aan te geven wat de bewerkingsafstand is tussen twee strings. Maar aangezien dit werkt voor alle soorten sequenties kunnen we dit ook gebruiken om het aantal nodige bewerkingen te vinden om twee routes in elkaar om te zetten. De Hamming distance is hierbij de bovengrens van de Levenshtein distance maar, we kunnen mogelijks lager gaan en een meer accurate afstand krijgen. De Levenshtein distance heeft echter wel een complexiteit van $O(n^2)$ waar de Hamming distance een complexiteit heeft van $O(n)$. Wanneer we de Levenshtein distance willen gebruiken voor de biased fitness is dit enkel een aanpassing in $d_H(P_1, P_2)$ de formule voor de biased fitness zelf blijft dezelfde.

We zien dat de resultaten voor de Levenshtein distance en de Hamming distance bijna niet verschillen, dit aangezien de Hamming distance al goed genoeg is als diversiteitsmetriek. Aangezien de diversiteit bepaalt dient te worden tussen alle paren in de populatie om de dichtste N oplossingen te vinden voor N_{close} is de biased functie echter een zware operatie met totale complexiteit voor het berekenen van de diversiteit: $O(p^2n^2)$ met n het aantal locaties en p de grootte van de populatie, hierdoor zien we voor grote VRPs dat het gebruiken van Levenshtein distance niet meer haalbaar is en door de lage winst die we ermee krijgen dus niet de moeite. Daarom gebruiken we dus de Hamming distance voor het bepalen van de diversiteit. Hierdoor is de complexiteit voor het berekenen van de diversiteit $O(p^2n)$

Ook door onze manier van population management en de random restarts behouden we diversiteit op lange termijn op de populaties aangezien hierdoor telkens per level in de hiërarchie oplossingen samen komen die op een andere manier bekomen zijn en nog nooit samen in een populatie gezeten hebben.

4.4.3 Optimalisaties en aanpassingen aan ADC biased fitness functie

Bij het veranderen van de populatie door het invoegen van een nieuwe oplossing kan in principe elke biased fitness functie van elke oplossing veranderen aangezien de diversiteit ten opzichte van de n dichtste buren bepaald wordt en dit kan veranderen door een nieuwe mutatie. We kunnen echter door gebruik te maken van dynamisch programmeren de fitness van elke oplossing en de Hamming distance tussen alle paren opslaan. Hierdoor moet bij het invoegen van een nieuwe oplossing enkel de paren met de oude oplossing vervangen worden door de nieuwe oplossing, als we de biased fitness nodig hebben

kunnen we deze in constante tijd opvragen door de fitness en Hamming distances op te halen van de oplossing en de formule toe te passen. Het berekenen van de biased fitness is hierdoor $O(1)$ en bij het toevoegen van een nieuwe oplossing is de diversiteit tabel aanpassen $O(pn)$ en de nieuwe fitness bepalen $O(n)$, aangezien de populatie p altijd een vaste grootte heeft is het invoegen van een nieuwe oplossing in de populatie dus $O(n)$ geworden.

Een andere aanpassing die we kunnen doen om de biased fitness beter regelbaar te maken en makkelijker te berekenen is om een nieuwe variabele dp te introduceren die het percentage voorstelt waarmee we de diversiteit laten doorwegen. Onze formule voor de biased functie wordt de volgende:

$$BF(P) = (1 - dp) \cdot fitness(P) + dp \cdot fitness(P) \cdot \Delta P$$

Hiermee kunnen we een percentage van de fitness laten afhangen van de diversiteit, wat het makkelijk maakt om over na te denken. Aangezien ΔP ook tussen 0 en 1 ligt kunnen we hiermee ook rechtstreeks fitness vergelijken, we weten namelijk dat $BF(P) \leq fitness(P)$. Dit heeft enkele voordelen voor de performantie: zo kunnen we nu door enkel de fitness te bepalen al zien wanneer we geen betere oplossing zullen bekomen doordat onze biased fitness maximaal gelijk zal zijn aan de fitness. Hierdoor kunnen we nog voor de diversiteit te berekenen al enkele slechte oplossingen snoeien die zeker geen survivor zouden worden en hierdoor de update stap van de diversificatie achterwege laten. Ook kunnen we de beste oplossing met de hoogste fitness voorstellen met de normale fitness zodat deze altijd bovenaan de populatie zal staan en niet na verloop van tijd eruit kan vallen door andere meer diverse oplossingen, hierdoor zijn we zeker dat de beste oplossing met de hoogste fitness altijd bovenaan zal staan. Het toevoegen van een oplossing in onze populatie en het bepalen van de biased fitness wordt gedaan zoals in algoritme 4. Hierbij is de population een gesorteerde lijst, en wanneer we *add* uitvoeren zal dit het element toevoegen op de plaats die de sortering behoud. De I stelt in deze algoritmes de nieuwe oplossing voor die toegevoegd dient te worden, en $diversityMatrix$ stelt een (p x p) matrix voor die de diversiteit tussen de paren van oplossingen in de populatie bijhoudt. De id property van de oplossing I, is een unieke waarde tussen 0 en p voor een population entry. Aangezien we mutaties in-place doen op de slechtste oplossing, moeten we deze dus enkel aan het begin van het algoritme instellen en worden deze hierna automatisch bijgehouden. Verwar de id dus niet met de index in de *population* array. Meer info hierover staat in de implementatie uitleg.

4.5 Fitness functie

Ons algoritme is een vrij generiek veelzijdig mutatie algoritme dat snel en gericht mutaties kan uitvoeren op oplossingen van PDSVRP. Hierdoor is het vrij generiek en zijn we dus niet gelimiteerd door één enkele fitness functie. Initieel werd het gebouwd om leveringen/uur te maximaliseren, maar de heuristieken en mutaties zullen over het algemeen gewoon slimme routes proberen construeren. Wanneer we dus andere fitness functies inpluggen kunnen we dus evengoed naar deze optimaliseren. Dit heeft natuurlijk ook zijn limieten, maar zolang de fitness functie efficiënte routes verwacht zal ons algoritme hiernaar kunnen optimaliseren. Daardoor kunnen we dus ook fitness functies voor makespan, minimale kost, ... inpluggen en hiernaar optimaliseren.

De uiteindelijk fitness functie waarvoor we gekozen hebben is makespan geworden aangezien deze het meeste gebruikt werd in de literatuur en ons dit ook een goede logische doelfunctie leek voor het probleem. Enkel de makespan gebruiken fitness functie is echter enorm inefficiënt, hierbij telt namelijk enkel de langste route mee in de fitness functie en de rest is vrij irrelevant. Wanneer we dus goede mutaties doen op andere routes dan de langste route zal dit niet weerspiegeld worden in de fitness functie en dus genegeerd worden. Zeker bij ons algoritme is dit een probleem aangezien we niet gericht werken maar eerder een grote hoeveelheid slimme snelle mutaties uitvoeren. We willen dus een fitness functie die elke verbetering in de oplossing reflecteert zodat we elke route tegelijk efficiënter maken.

Algorithm 4 Voeg oplossing I toe aan de populatie

Input: I, $diversityMatrix$, $population$

$I.fitness \leftarrow fitness(I)$;

$I_{worst} \leftarrow population[p-1]$;
if $I.fitness < I_{worst}.BF$ **then**
  // Will never survive
  $population.add(\{I.fitness, I\})$;
  return *population*;
end if

for $P \in population$ **do**
  $diversityMatrix[max(I.id, P.id)][min(I.id, P.id)] = d_H(I, P)$;
end for

$I_{best} \leftarrow population[0]$;
if $I.fitness > I_{best}.fitness$ **then**
  $population.add(\{I.fitness, I\})$;
  // score of best solution = fitness, so calculate BF of old best
  $I_{best}.BF \leftarrow BF(I_{best})$;
  $population.remove(\{I_{best}.fitness, I_{best}\})$;
  $population.add(\{I_{best}.BF, I_{best}\})$;
  return *population*;
end if
$I.BF \leftarrow BF(I)$;
$population.add(\{I.BF, I\})$;
return *population*;

Als andere routes ook efficiënter zijn kunnen we namelijk ook sneller de makespan verbeteren door de minder efficiënte routes locaties te laten overdragen aan de efficiënte kortere routes.

De oplossing die we hiervoor gemaakt hebben is een fitness functie die tegelijk de efficiëntie als de makespan probeert te optimaliseren. We proberen het aantal leveringen/uur te maximaliseren en de makespan te minimaliseren. De leveringen/uur van een route drukken we uit als:

$$ph(r) = \frac{|r|}{dist_r}$$

De fitness functie waarnaar we maximaliseren is:

$$fitness(r) = \frac{|r|}{dist_r + makespan}$$

Hiermee wordt zowel de leveringen/uur als de makespan geoptimaliseerd. Er valt echter wel nog een trad-off te maken in sommige gevallen aangezien we hiermee uiteindelijk de makespan willen minimaliseren kan het hier wel gebeuren dat soms efficiëntere routes met slechtere makespan kunnen beter scoren dan enkel de makespan. We kunnen hierdoor de makespan belangrijker maken door er een factor voor te zetten we bekomen dan:

$$fitness(r) = \frac{|r|}{dist_r + f \cdot makespan}$$

Hierbij is f een factor die aangeeft hoe belangrijk de makespan is. In principe kunnen we deze zo hoog maken zoals we willen, dat zal er enkel voor zorgen dat de fitness bijna uitsluitend door makespan bepaald zal worden. Maar we hebben gemerkt dat vooral in het begin goede routes goed belonen zorgt voor betere eindoplossingen. De waarde voor f dat we hierdoor vaak gebruiken is rond de 5, maar vaak maakt het geen groot verschil. Naar het einde toe willen we uiteindelijk wel de maximale makespan dus zorgt een hoge f wel dat we bijna gegarandeerd de beste makespan zullen geven die we gevonden hebben.

Twee dingen die we nog hebben opgelost in onze fitness functie is de constraints van wagen capaciteit en drone toegankelijke locaties. We hebben dit geïmplementeerd als soft contraints die een grote penalty geven aan de eindoplossing. Daardoor moeten we geen checks of restricties opleggen aan mutaties, maar kan alles door de fitness geregeld worden. We maken deze penalty's vrij hoog zodat de uiteindelijke oplossing bijna nooit constraints zal overschrijden. Naar het einde toe maken we de penalty's ook zo hoog dat enkel oplossingen die geen constraints overtreden worden als eindoplossing gegeven zullen worden. Deze soft contraints hebben als voordeel dat we soms bij grote verbeteringen die een constraint lichtjes overtreedt toch kunnen toelaten met een penalty en deze penalty kan later weggewerkt worden. Zo laten we meer grote mutaties toe.

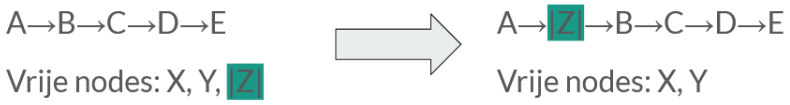
4.6 Mutaties

Mutaties zijn de bouwstenen waaruit het genetisch algoritme bestaat. In vele papers zoals Vidal et al [66] worden ook mutaties gebruikt, maar vaak vallen deze onder de noemer van localsearch. In bijna alle oplossingsmethodes die localsearch toepassen met een meta-heuristiek worden mutaties toegepast. In essentie zijn dit operaties die een verandering brengen in een oplossing met als resultaat een nieuwe oplossing. In ons geval koppelen we de mutaties los van localsearch en werken we meer op een klassieke genetische manier waarbij cross-over en random mutatie in random volgorde en hoeveelheden voorkomen. Hierbij hebben mutaties en cross-overs een random percentage als gewicht toegewezen dat aangeeft wat de kans is dat deze uitgevoerd wordt in een mutatie stap. Op deze manier kunnen we de frequentie van bepaalde mutaties finetunen. Door de nood aan snelle convergentie is er nood

aan heuristieken voor de meeste mutaties. Bij volledig random mutaties zijn de resultaten vaak ook redelijk goed, maar vaak kwamen we ook vast te zitten voor lange tijd in slechte paden, en was de tijd tot convergentie zeer lang. Heuristieken tonen hier een goede oplossing, zolang de heuristieken efficiënt genoeg zijn en geen grote limitatie geven op de nuttige zoekruimte is dit geen probleem. Maar het doel van de heuristieken is zoveel mogelijk de onnuttige zoekruimte te snoeien, en ons vooral de nuttige zoekruimte te laten doorzoeken. In de literatuur worden mutaties ook soms moves genoemd, vaak wordt gesproken over volgende varianten bij VRP, maar ook in algemene genetische algoritmes: swap, relocate en 2-opt. Bij swap, worden 2 locaties in de oplossing omgedraaid. Bij relocate wordt de positie van een locatie op een andere plaats geplaatst in de oplossing, dit kan een andere plaats in dezelfde route zijn maar ook in een nieuwe (drone) route. 2-opt is een localsearch move ontworpen door Croes et al. [12] waarbij het doel is kruisingen in een route met zichzelf ongedaan maken. Ook voor TOP en PTP zien we niet veel speciale mutaties terugkomen, vaak de 1-move en swap-move, welke eigenlijk equivalent zijn aan relocate en swap. In een survey over TOP van Vansteenwegen et al. [63], waren ook alle verschillende soorten moves allemaal verwerkt in onze mutaties, maar vaak slechts voor 1 node terwijl wij dit voor meerdere nodes toestaan.Ook mutaties zoals cross en cross-over, die we later zullen uitleggen, waren hier niet gebruikt. Maar wel moves zoals add en remove.

Met de moves swap en relocate kan men in principe de volledige oplossingsruimte doorzoeken, maar wij hebben ons hier echter niet door beperkt. We hebben nog een aantal andere mutaties gedefinieerd zodat we een aantal goede tools hebben om manipulaties te doen op onze oplossingen. We bespreken nu de verschillende mutaties die we gedefinieerd hebben en de heuristieken die we op deze mutaties toegepast hebben.

4.6.1 Add



Figuur 4.5: Add mutatie

Mutate add (figuur 4.5) is een mutatie operator waarbij een vrije locatie toegevoegd wordt aan een route. Dit kan gezien worden als een relocate mutatie van een vrije locatie in een route. We hebben echter gekozen om de add mutatie ook meerdere locaties in één keer te laten toevoegen, deze worden allemaal op dezelfde plaats in de route toegevoegd. Op deze manier kunnen we routes makkelijk uitbreiden. De keuze voor meerdere locaties te laten toevoegen in één mutatie is er gekomen om eilanden van nodes sneller te kunnen toevoegen aan een route. Wanneer we bijvoorbeeld slechts een locatie toevoegen van een eiland kan de fitness van de route achteruitgaan, maar als we meerdere van een eiland in één keer kunnen toevoegen kan dit eiland van locaties opgenomen worden in de route met een toename van de fitness.

Er zijn enkele methodes om de beste plaats te bepalen waar een node in een route toegevoegd kan worden. We kunnen een random positie in een random route nemen, en hier een random locatie toevoegen, dit houdt de volledige zoekruimte open, maar zal ook zeer inefficiënte mutaties uitvoeren. De nuttige zoekruimte waartoe we ons kunnen beperken is namelijk vrije locaties op plaatsen in routes die dicht bij deze vrije locatie liggen. Er zijn verschillende manieren waarop we dit kunnen bepalen, maar een van de belangrijkste criteria hier is hoe dicht deze bij elkaar dienen te liggen. Dit is wederom een afweging tussen exploratie en efficiëntie. Een korte afstand zou enkel goede nodes toevoegen, maar een verre afstand houdt de zoekruimte breder en kan op termijn voor betere oplossingen zorgen en grotere veranderingen teweegbrengen.

Een metriek die ons hierbij kan helpen de afstand te bepalen is de gemiddelde locatie afstand van een route of oplossing μd, deze geeft aan wat de gemiddelde afstand is op onze route tussen twee opeenvolgende locatie.

$$\mu d = \frac{dist_r}{|r|}$$

Hierbij is $dist_r$ de afstand van een route, of de totale afstand die een koerier moet afleggen als deze de route uitvoert vanuit de depot. $|r|$ is gelijk aan het aantal locaties dat op de route r liggen.

Als we de gemiddelde afstand μd hebben berekend weten we een goeie radius waarin we locaties kunnen zoeken om toe te voegen aan een route die een grote kans op verbetering geven. Wanneer we een fitness functie gebruiken zoals de ratio locaties/uur, makespan of totale afstand, hangt deze namelijk zeer hard samen met μd. Bij locaties/uur, de fitness functie die wij hanteren voor DVRPD, is er zelfs een 1-op-1 relatie. We kunnen ook voor het toevoegen in een route perfect bepalen welke nodes een verbetering kunnen geven aan de locaties/uur van een route. Dit door bij elke opeenvolgende locaties in de route een ellips te construeren met deze punten als brandpunten en als lengte $a = \mu d + dist(p_1, p_2)$. Als we in deze ellips een vrije locatie vinden weten we dat deze toevoegen tussen de punten een directe verbetering zal geven aan μd van de route en dus ook aan de locaties per uur van die route. Het kan echter wel dat de totale μd wel niet verbeterd omdat de locatie efficiënter door een drone gedaan kon worden. Al deze ellipsen construeren en de punten die erin vallen vinden is computationeel intensief, daarom kunnen we een vereenvoudiging maken die voor ons doel goed genoeg is en een pak efficiënter. We kunnen vanuit een vrije locatie een cirkel construeren met als straal μd en de punten bepalen binnen de cirkel die op een route liggen. We kunnen nu de vrije locatie proberen invoegen voor of na de punten in deze cirkel die op een route liggen en is de kans op een verbetering al een pak groter. De pseudocode voor het uitvoeren van dit algoritme wordt gegeven in algoritme 5.

Een andere heuristiek die gebruikt kan worden is met het gebruik van cirkelsectoren. We kunnen de cirkelsector waarin een route valt met als middelpunt de depot bepalen met poolcoördinaten. Hierna kunnen we ook makkelijk checken of een punt in deze cirkelsector ligt met behulp van de poolcoördinaten van het punt. Op deze manier kunnen we enkel punten toevoegen aan de route die binnen de cirkelsector liggen, met mogelijks een marge aan de uiteinden.

Een andere meer globale heuristiek heeft te maken met het efficiënt inplannen van de vrije locaties aan drones. Drones hebben namelijk het beste locaties die dicht bij de depot liggen, dit omdat deze elke keer terug van de depot vertrekken, in tegenstelling tot wagens die vanaf de vorige locatie starten. Zo kan een route gemaakt worden naar een verre locatie die zeer efficiënt is, maar een drone zal altijd de verre afstand moeten overbruggen naar deze locatie. Daarom is het over het algemeen goed om verre locaties op te nemen in een route van een wagen, en dichte locaties door drones te laten doen. Dit is ook goed voor de beperkte range van drones. Daarom is in een add mutatie vrije locaties nemen ver van de depot vaak het beste voor de fitness van het volledige systeem.

Optimalisaties in de implementatie

Het zoeken van de neighbors van een vrije locatie kan een vrij intensieve operatie zijn, zeker wanneer we dit naïef doen kan dit tot een complexiteit van O(n) gaan voor het bepalen van de neighbors. Aangezien we willen dat de mutaties snel verlopen en elke keer de neighborhood nodig hebben kunnen we beste een gespecialiseerde datastructuur verkiezen die geoptimaliseerd is hiervoor. We kunnen binning van de locaties toepassen volgens hun coördinaten, zo kunnen we de volledige coördinaat ruimte die gaat van $min - x$ tot $max - x$ en van $min - y$ tot $max - y$ opdelen in een grid van (m x n). Elke bin bevat alle locaties in een tegel van de coördinaat ruimte. Als we van een locatie de neighbors willen kunnen we de locaties geven in dezelfde bin en de omringende bins. Er dient hier wel een afweging gemaakt te worden hoe groot we ons (m x n) rooster nemen. Een meer elegante oplossing is het

Algorithm 5 Add mutatie heuristiek

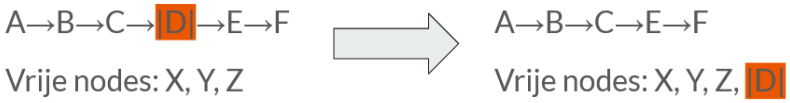
$fl \leftarrow$ random free location;
$neighbours \leftarrow$ All location in a proximity of μd;

if no location in neighbours in a route **then**
  return NULL;
end if

$insertPos \leftarrow$ random location of neighbours in a route;
$fls \leftarrow$ all free locations in neighbours;
$addAmount \leftarrow$ random number between 1 and $|fls|$;
$newSeq \leftarrow$ sequence of $addAmount$ random locations out of fls;
insert $newSeq$ on $insertPos$;

gebruiken van het opslaan van de locaties in een k-d tree [6], dit is space-partitioning datastructuur die punten kan organiseren in de ruimte waarbij operaties zoals het opvragen van de dichtste buren gereduceerd worden naar gemiddeld $O(log\, n)$. Het opvragen van buren met behulp van binning per range coördinaten is echter $O(1)$ en goed genoeg voor ons doeleinde, daarom gebruiken we deze in de reference implementatie.

4.6.2 Remove



Figuur 4.6: Remove mutatie

Bij de remove mutatie halen we een locatie uit een route en wordt deze een vrije locatie die in geen enkele route zit (figuur 4.6). We kunnen deze mutatie ook zien als een relocate mutatie van een locatie van een route naar een vrije locatie. Ook hier hebben we gekozen het toe te staan meerdere locaties in één keer te verwijderen. Om analoge reden als bij de add mutatie laten we dit toe. Vaak zal er pas verbetering mogelijk zijn als een eiland van nodes removed kan worden. Ook hier kunnen we enkele heuristieken definiëren om de meest nuttige locaties een hogere probabiliteit te geven. We kunnen makkelijk in lineaire tijd vinden welke locaties het slechtste zijn in het geheel van alle routes. Dit kunnen we doen door een eenvoudig algoritme die de routes doorloopt en kijkt voor 3 locaties l_i, l_{i+1} en l_{i+2} wat de toegevoegde afstand Δd is door l_{i+1} met volgende formule:

$$\Delta d_{i+1} = dist(l_i, l_{i+1}) + dist(l_{i+1}, l_{i+2}) - dist(l_i, l_{i+2})$$

We kunnen makkelijk de locatie met de grootste toegevoegde afstand weglaten uit de route en een vrije locatie maken. De pseudocode van deze heuristiek wordt gegeven in algoritme 6. Op een gelijkaardige manier kunnen we dit ook doen voor het verwijderen van meerdere locaties, dan breiden we onze som gewoon uit naar de n te verwijderen locaties. Hierna kijken we naar het verschil in afstand tussen het te verwijderen stuk en het verbinden van de uiteinden van dit stuk. Mochten we telkens het deel met de hoogste Δd verwijderen dan wordt onze mutatie zeer deterministisch, daarom voegen we best nog wat randomness toe. Het is namelijk zeker niet gegarandeerd dat deze remove operatie een verbetering zal geven. Want mochten we telkens de beste remove operatie nemen, maar deze zorgt voor geen verbetering, dan zouden we bij een volgende remove mutatie net dezelfde mutatie doen en nooit verbetering halen. Daarom sorteren we de mogelijkheden volgens hun Δd waarden en kiezen een

random mutatie met meer kans voor de mutaties met hoge Δd. Aangezien we toch elke Δd berekenen heeft dit geen verlies in performantie.

Een andere heuristiek die mogelijks tot betere resultaten kan komen is het verwijderen van locaties die dicht/dichter bij andere routes liggen. Als we zien dat een locatie volledig omgeven is door locaties van een andere route is het soms slimmer deze te verwijderen zodat deze door de andere route opgenomen kan worden.

Ook kunnen we hier hetzelfde argument geven over de afstand tot de depot, zo verwijderen we liever locaties dicht bij de depot dan ver van de depot. Met dezelfde redenering als gegeven bij de add mutatie.

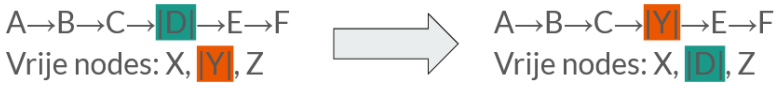
Algorithm 6 Remove mutatie heuristiek

$route \leftarrow$ random route;
$removeAmount \leftarrow$ random number between 1 and $|route|$;
$bestRemove \leftarrow \varnothing$;

for i between 0 and $|route| - removeAmount$ **do**
  $l_s \leftarrow route[i]$;
  $l_{s+1} \leftarrow route[i+1]$;
  $l_e \leftarrow route[i + removeAmount]$;
  $l_{e+1} \leftarrow route[i + removeAmount + 1]$;
  $\Delta d = dist(l_s, l_{s+1}) + dist(l_e, l_{e+1}) - dist(l_s, l_{e+1})$

  if $\Delta d > bestRemove$ **then**
    $bestRemove \leftarrow \Delta d$
  end if
end for
Remove the $removeAmount$ locations after the location that gave $bestRemove$;

4.6.3 Swap-free



Figuur 4.7: Swap Free mutatie

Bij de swap-free mutatie verwisselen we een locatie uit een route met een vrije locatie (figuur 4.7). Zoals de naam al doet vermoeden is dit een swap operatie. Deze operatie zal altijd gebeuren tussen 1 locatie uit een route en 1 vrije locatie. Voor het verwisselen van locaties tussen routes hebben we de cross-over mutatie gedefinieerd die later uitgelegd wordt. Ook hier zijn enkele heuristieken mogelijk. De meest voor de hand liggende heuristiek is deze waarbij we enkel locaties swappen die dicht bij elkaar liggen. Hierbij kunnen dezelfde strategie gebruiken zoals bij de add heuristiek. Deze zal dus voor een vrije locatie vooral naar locaties proberen zoeken die dichter dan de gemiddelde locatie afstand Δd liggen. We kunnen ook opnieuw per opeenvolgende locaties in de routes kijken welke vrije locaties dicht liggen, maar dit is ook identiek aan de heuristiek besproken bij de add mutatie. De aangepaste heuristiek voor de swap-free wordt gegeven in algoritme 7.

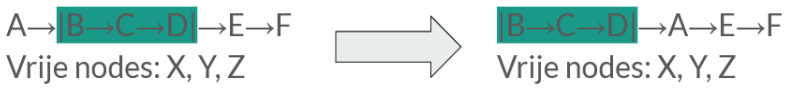
Algorithm 7 Swap-free mutatie heuristiek

$fl \leftarrow$ random free location;
$neighbours \leftarrow$ all location in a proximity of μd;

if no location in neighbours in a route **then**
  return ;
end if

$location \leftarrow$ random location of neighbours in a route;
$swap(fl, location)$;

4.6.4 Swap-pos



Figuur 4.8: Swap position mutatie

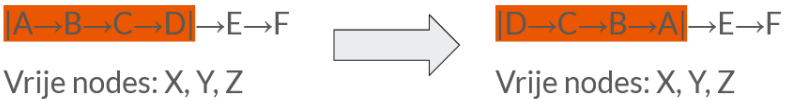
Bij swap-pos veranderen we de positie van een locatie in de route (figuur 4.8). Deze mutatie valt ook te classificeren onder de relocate mutaties. Bij deze mutatie valt de positie steeds in dezelfde route. We hebben echter een uitbreiding gemaakt waarbij we n opeenvolgende locaties van positie verwisselen. Zo kunnen we grotere delen in een route in 1 keer verschuiven.

Een heuristiek voor de swap-pos mutatie is in principe niet moeilijk te vinden aangezien alle mogelijke opties waarin we een sequentie kunnen positioneren snel gecheckt is. We kunnen dus voor een random sequentie waarvan we de positie willen aanpassen alle mogelijke posities overlopen en de afstand van de start en end node tot de plaats van insertion bekijken en degene kiezen die de beste insertion plaats blijkt te zijn. De pseudocode wordt gegeven in algoritme 8.

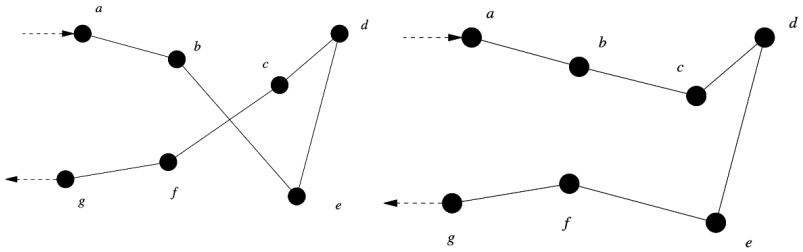
Algorithm 8 Swap-pos muatie heuristiek

$route \leftarrow$ random route;
$pos \leftarrow$ random pos in $route$
$seqLen \leftarrow$ random number between 1 and $|route| - pos$;
$bestInsertPos \leftarrow \varnothing$
$l_s \leftarrow route[pos]$;
$l_e \leftarrow route[pos + seqLen - 1]$;
for i between 0 and $|route| - seqLen$ **do**
  if $i = pos$ **then**
    $i \leftarrow i + seqLen$
  end if
  $l_{s-1} \leftarrow route[i - 1]$;
  $l_{e+1} \leftarrow route[i + seqLen + 1]$;
  $\Delta d = dist(l_{s-1}, l_s) + dist(l_e, l_{e+1}) - dist(l_{s-1}, l_{e+1})$

  if $\Delta d > bestInsertPos$ **then**
    $bestInsertPos \leftarrow \Delta d$
  end if
end for
insert the sequence between l_s and l_e after the location that gave $bestInsertPos$;



Figuur 4.9: 2-OPT mutatie voorbeeld illustratie



Figuur 4.10: 2-OPT mutatie gevisualiseerd

4.6.5 2-OPT

Bij de 2-OPT mutatie proberen we lussen in routes op te sporen en aan te passen zodat er geen lus meer is. Dit omdat kruisende routes altijd een langere afstand zullen hebben dan hun niet kruisende variant, dit wordt geïlustreerd op figuur 4.10. Het oplossen van zo een lus doen we door het selecteren van de sequentie van opeenvolgende locaties in de lus, deze om te draaien en vervolgens terug toe te voegen op dezelfde locatie oals op figuur 4.9. Deze mutaties werd beschreven door Croes et al. [12]. Wanneer we namelijk een 2-OPT operatie kunnen doen op een route die zichzelf kruist halen we de kruising weg en wordt de resulterende route altijd korter (figuur 4.10).
Als we bij de mutatie die we uitvoeren bij 2-OPT kijken naar het verschil in afstand zien we dat de afstand van het deel voor en na de sequentie hetzelfde blijft, ook de afstand van de sequentie zelf blijft behouden aangezien we enkel de sequentie omgekeerd doorlopen en de afstanden symmetrisch zijn. De enige bogen die aangepast worden in de route zijn de bogen voor en na de sequentie. Deze worden namelijk omgedraaid. Als we dus willen kijken welke 2-OPT operaties een verbetering zullen geven in de route moeten we dus enkel checken of de bogen voor en na de sequentie omdraaien een kortere afstand heeft dan de originele.

$$\Delta d = dist(l_{s-1}, l_e) + dist(l_s, l_{e+1}) - dist(l_{s-1}, l_s) - dist(l_e, l_{e+1})$$

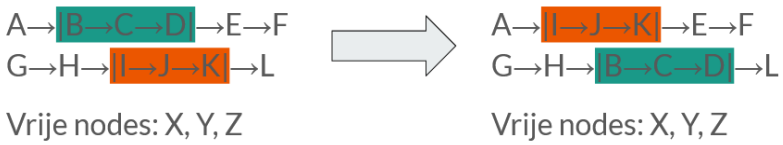
Waarbij Δd het verschil is in afstand na 2-opt, als dit verschil negatief is dan hebben we dus een kortere route. l_s is hierbij de start locatie van de sequentie en l_e is het einde van de sequentie. Wanneer we over alle koppels van bogen in de route lopen die niet met elkaar verbonden zijn en we berekenen Δd dan kunnen we alle mogelijke verbeteringen met 2-opt uitvoeren. We voeren echter enkel de eerstgevonden verbetering uit, hierna geven we de mutatie door. Over deze bogen loopen heeft een complexiteit van $O(n^2)$.

In algoritme 9 zien we de werking van de 2-OPT mutatie. De loops in de pseudocode gaan over alle bogen van de route die niet aan elkaar verbonden zijn. Op deze manier overlopen we alle koppels en dus alle mogelijke plaatsen waar we 2-OPT kunnen toepassen. We stoppen wanneer we een 2-OPT operatie gevonden hebben die een verbetering geeft en voeren deze ook uit. Als we geen verbetering vinden proberen we een andere mutatie.

Algorithm 9 De 2-OPT mutatie

```
route ← random route
for edge e1 in route do
  for edge e2 that comes after e1 in route do
    if e1 and e2 do not share any node then
      l11, l12 ← nodes of e1
      l21, l22 ← nodes of e2
      Δd ← dist(l11, l21) + dist(l12, l22) − dist(l11, l12) − dist(l21, l22)
      if Δd < 0 then
        reverse part between e1 and e2
        replace e1 by an edge between l11 and l21
        replace e2 by an edge between l12 and l22
        return route
      end if
    end if
  end for
end for
```

4.6.6 Cross-over mutatie



Figuur 4.11: Cross-over mutatie

We kunnen binnen een oplossing zelf ook een cross-over mutatie definiëren, niet te verwarren met de cross-over operator die tussen verschillende oplossingen werkt en hierna besproken wordt. Bij de cross-over mutatie nemen we willekeurig 2 routes uit een oplossing en definiëren waar in de routes we een cross-over willen doen. Wanneer we de insertion point en lengte gekozen hebben verwisselen we de stukken van de routes, zoals in een normale cross-over (figuur 4.11). We hebben hier gekozen om met een losse variant van cross-over te werken waarbij de insertion index en lengte van het verwisselde stuk niet hetzelfde hoeven te zijn. We kunnen dus stukken van routes van verschillende lengte en positie in de route verwisselen.

We hebben ook voor deze mutatie enkele heuristieken die we kunnen gebruiken. Allereerst is het vrij logisch dat we vooral goede oplossingen zullen vinden als het insertion point van de ene route dicht ligt bij de start van de getransfereerde sequentie, en aan de andere kan dat het einde ook dicht ligt. Een mogelijke methode die dit mogelijk maakt is een heuristiek die voor een gegeven route, insertion point en lengte kijkt naar de nodes rond het startpunt l_s en eindpunt l_e die van een andere gemeenschappelijke route zijn. Als we zo een paar van nodes vinden l_sn en l_en, bij de start en het einde van de sequentie kunnen we het stuk tussen l_s en l_e omwisselen met de sequentie tussen l_{sn} en l_{en}. De lengte van deze sequentie kan wel serieus verschillen, wat zelden voor een verbetering zal leiden, hier ligt de grootste zwakte van deze methode. Deze pseudocode van deze heuristiek is te vinden in algoritme 10.

Een andere heuristiek is een die voor twee random routes kijkt welke punten het dichtste bij elkaar liggen en de transfer met de sequentie tussen die punten doen.

Algorithm 10 cross-over mutatie heuristiek

$r1 \leftarrow$ random route
$pos1 \leftarrow$ random locatie in route
$pos2 \leftarrow$ random locatie in route after $pos1$
$neighbors1 \leftarrow$ all location in a proximity μd of $pos1$;
$neighbors2 \leftarrow$ all location in a proximity μd of $pos2$;
Find a location $pos3$ in $neighbors1$ and $pos4$ in $neighbors2$ that are of the same route but different from $r1$;

if We didn't find a pos3 and pos4 **then**
  return NULL;
end if

swap the sequence between $pos1$ and $pos2$ with the sequence bestween $pos3$ and $pos4$

Een andere heuristiek is een die sequenties gaat zoeken die omgeven zijn door locaties die in een andere route liggen, als we deze hebben gevonden doen we een cross-over met deze route.

Over het algemeen is het ook een goede heuristiek om de lengtes van de sequenties zo gelijk mogelijk te houden.

4.6.7 Cross

Een cross mutatie is een soort cross-over mutatie, maar slechts op 1 plaats wordt de boog getransfereerd in plaats van op 2 plaatsen. Dit dient onder andere om kruisende verschillende routes weg te werken, maar kan ook voor andere dingen handig zijn. Een andere interessante eigenschap die we hier kunnen bovenhalen is dat we vrij makkelijk onze routes kunnen omdraaien, we kunnen hierdoor tussen twee bogen van verschillende routes 2 verschillende crosses definiëren. Als we $l_1 1$ en $l_1 2$ hebben van boog 1 en $l_2 1$ en $l_2 2$ van boog 2. Kunnen we de bogen als volgt veranderen $edge(l_1 1, l_2 1)$ en $edge(l_1 2, l_2 2)$, maar ook $edge(l_1 1, l_2 2)$ en $edge(l_1 2, l_2 1)$. Wanneer we kiezen voor $edge(l_1 1, l_2 1)$ en $edge(l_1 2, l_2 2)$ moeten we echter een route omdraaien aangezien we anders twee keer in dezelfde richting gaan en in een deadlock gaan zitten op de getransfereerde boog.

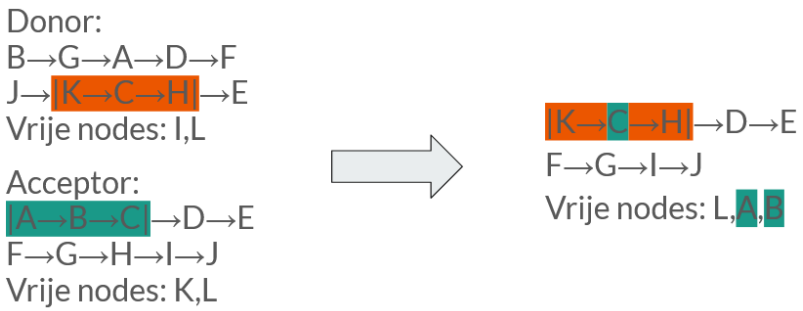
We kunnen ook hiervoor een variant van 2-OPT definiëren 2-OPT* die alle kruisen uit onze VRP kan halen, deze loopt over alle combinaties van bogen over alle routes wat een complexiteit geeft van $O(n^2)$ als n gelijk is aan het aantal bogen in het VRP. Waar 2-opt een intra-route mutatie is, is 2-OPT* de inter-route mutatie variant ervan, deze gaat kruisingen van 2 verschillende routes ongedaan maken. De pseudocode van dit algoritme wordt gegeven in algoritme 11.

Deze inter-route 2-OPT mutatie zal echter niet enkel kruisen oplossen zoals bij de normale 2-OPT, maar kan ook elders vereenvoudigingen doen. Zo kan die bijvoorbeeld de routes ook beter balanceren of op andere manieren die efficiëntie ervan bevorderen. Een andere heuristiek mogelijk voor inter-route 2-OPT is bogen die dicht bij elkaar liggen proberen draaien.

Algorithm 11 De inter-route 2-OPT mutatie

```
for route r1 in routes do
  for edge e1 in r1 do
    for route r2 in routes\{r1} do
      for edge e2 in r2 do
        l11, l12 ← nodes of e1
        l21, l22 ← nodes of e2
        Δd1 ← dist(l11, l21) + dist(l12, l22) − dist(l11, l12) − dist(l21, l22)
        if Δd1 < 0 then
          reverse r1
          replace e1 by edge(l21, l11)
          replace e2 by edge(l12, l22)
          return route
        end if
        Δd2 ← dist(l11, l22) + dist(l12, l21) − dist(l11, l12) − dist(l21, l22)
        if Δd2 < 0 then
          replace e1 by edge(l11, l22)
          replace e2 by edge(l21, l12)
          return route
        end if
      end for
    end for
  end for
end for
```

4.7 Cross-over



Figuur 4.12: cross-over voorbeeld

Er werd ook een cross-over gedefinieerd tussen twee oplossingen (figuur 4.12). Deze werkt gelijkaardig aan de cross-over mutatie, maar werkt tussen twee VRP oplossingen in plaats van twee routes. We stellen een oplossing aan als donor en de andere als acceptor, daarna gaan we een deel van een route van de donor inbrengen in een deel van een route van de acceptor. De overeenkomstige nodes die gebruikt werden worden uit de andere routes gehaald en de overschreven nodes worden als vrije nodes geplaatst.

De manier waarop we de cross-over definiëren is gelijkaardig aan deze in Vidal et al. [65] en wordt ook geïllustreerd op figuur 3.1. De manier waarop deze cross-over werkt gaat als volgt. Eerst bepalen we de 2 parents waarop de cross-over zal gebeuren. Een parent wordt de donor, de andere de acceptor. Hierna nemen we een random start van de cross-over s en een random lengte l in de donor. Dit deel

kan uit meerdere routes bestaan. We starten met het over kopiëren van dit deel, we houden een set bij van alle locaties aanwezig in dit deel. Hierna overlopen we de acceptor vanaf het begin en kopiëren alle nodes over die niet in deze set zitten, wanneer deze wel in de set zitten worden deze geskipt in de route. Wanneer we echter de start node s van onze cross-over tegenkomen in de acceptor, lopen we over de volgende l nodes, en kopiëren deze over als vrije nodes als ze niet in de set zitten en als ze er wel inzitten doen we niks aangezien deze al over gekopieerd zullen zijn uit de donor. Dit zijn de nodes waarover het deel van de donor gekopieerd wordt. Na deze l nodes gaan we terug naar de vorige manier van overlopen waarbij we dus de nodes niet in de set over kopiëren en degene wel in de set skippen. Het proces wordt uitgeschreven in algoritme 12.

Algorithm 12 cross-over

```
donor ← random solution out of population;
acceptor ← random solution out of population different from donor;
mutation ← empty solution;
seq ← get random sequence from the donor;
copy over seq into the mutation;
usedLocs ← set of locations in seq;

for route in acceptor do
  for location in route do
    if location is start of seq then
      insert seq;
      make overwritten locations not in usedLocs free locations;
      go to end of seq;
    end if

    if location in usedLocs then
      skip this location;
    else
      copy over location to mutation;
    end if
  end for
end for
```

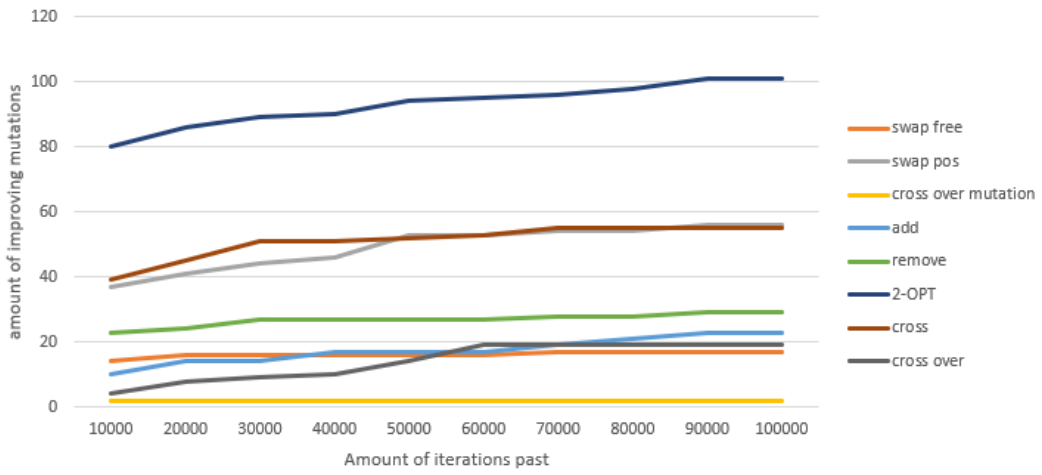
Een cross-over is een vrij intense operatie aangezien er heel wat bijgehouden en gekopieerd dient te worden, wat door de memory accesses een stuk trager kan zijn. Ook zal een cross-over zelden voor een directe verbetering zorgen, maar zit er vaak wel veel potentieel in deze oplossingen. Vidal et al. [65] loste dit als volgt op: na elke cross-over werd een localsearch process gestart waarbij we het lokale optimum zoeken van de oplossing na cross-over. Dit is echter ook een vrij dure operatie als we dit telkens blijven doen tot we geen verbetering meer vinden bij de localsearch zeker voor grote problemen wordt dit zeer intensief. Daarom kunnen we werken met een aantal iteraties mutaties n_r wat een repair stap voorstelt na een cross-over. Dit heeft dus als doel meer oplossingen na een cross-over in onze populatie te hebben en zo meer diverse oplossingen. Dit is echter niet in onze reference oplossing geraakt, maar is een mogelijke toekomstige verbetering die gedaan kan worden.

4.8 Analyze van de verschillende mutaties

Wanneer we kijken naar de performantie van de verschillende mutaties zien we soms grote verschillen in het de doeltreffendheid van de mutaties. We hebben ook enkele experimenten opgesteld voor het meten van de doeltreffendheid. Allereerst hebben we gekeken hoeveel keer een mutatie voor verbetering zorgt

bij een oplossing. We hebben dit gedaan door 100.000 iteraties uit te voeren van ons HGS algoritme met alle gewichten voor alle mutaties gelijk, zodat elke mutatie dus even veel uitgevoerd wordt, het percentage cross-overs tegenovers mutaties was 10%/90% en er zijn 7 mutaties wat neer komt op 12,5% van de iteraties dat een mutatie gedaan wordt. De makespan die we berekenen is volgens de makespan formule van Salue et al. [46], ondanks de kritiek die we erop zullen uiten is dit momenteel de meest gebruikte metriek. We hebben de tests gedaan op de X-139-k10 instantie van de CVRPLIB [71] met 5 drones en 5 wagens. En elke mutatie die een verbetering in de fitness functie te weeg bracht geteld doorheen het aantal iteraties. We hebben de testen 10 keer uitgevoerd en de gemiddelden genomen van de resultaten. De resultaten hiervan zijn te zien in figuur 4.13.

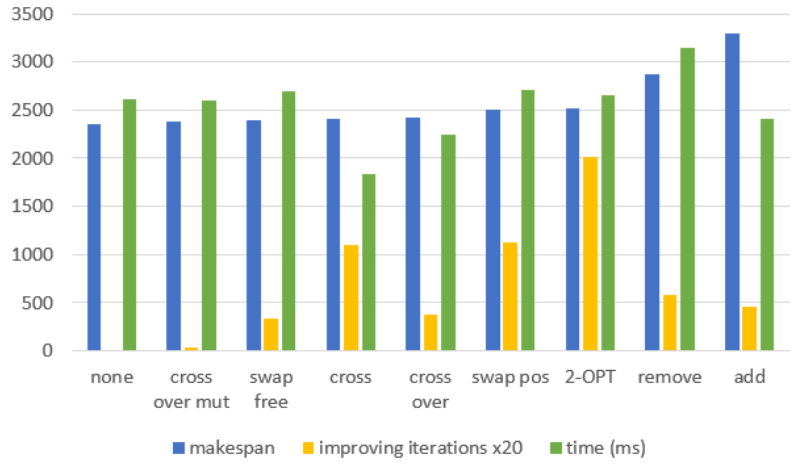
Figuur 4.13: Verbetering van mutaties over 100k iteraties



Op de grafiek in figuur 4.13 zien we allereerst dat het grootste deel van de improving mutaties al gedaan is de eerste 10k iteraties. Tijdens de eerste 10k iteraties zijn er 209 improving mutaties gebeurd en tijdens de resterende 90k iteraties slechts 93. Dit is enerzijds wel logisch aangezien we initieel veel ruimte tot verbetering is, maar na verloop van tijd hebben we een goede oplossing en is verbeteringen zoeken vrij moeilijk. We zien dit ook aan de afgeplatte curves. Er was echter wel veel verschil tussen de mutaties onderling, zo zagen we vooral vaak in de eerste 10k iteraties veel verschillen, dit omdat de eerste mutaties zeer vaak verbeteringen zullen geven en deze veel ruis kunnen introduceren. Vooral tussen 2-OPT en swap pos zagen we vaak dat deze ook andere verhoudingen aannamen. De mutaties die de meeste improvements geven zijn duidelijk 2-OPT, cross en swap-pos. Van 2-OPT en cross was dit echter wel te verwachten aangezien we hier ook de meeste rekentijd in stoppen om goede mutaties te kiezen. De enorm lage score voor de cross-over mutatie is ook wel zeer opvallend, dit geeft dus aan dat de gekozen heuristiek hiervoor niet goed werkt in ons systeem.

Een tweede experiment dat we gedaan hebben is het vergelijken van de oplossingen op X-n139-k10 met 100k iteratie en dezelfde configuratie als hierboven met de verschillende mutaties één voor één weggehaald. We draaien dus hetzelfde algoritme zonder een bepaalde mutatie en vergelijken hier de makespan, het aantal verbeterde oplossingen door de mutatie en de uitvoeringstijd. We hopen hierdoor een beeld te scheppen van hoe belangrijk de mutatie is voor het algoritme. De resultaten hiervan zijn te zien in figuur 4.14. We zien over het algemeen dat wanneer we mutaties met een hoger aantal improving iterations weglaten, we een grotere impact zien in onze makespan. Dit is ook wat onze intuïtie ons zou vertellen. Enkel bij de add en remove operatie is dit niet volledig in lijn, maar ook hier is een logische verklaring voor. Add en remove regelen vooral de aantal locaties per route, dit is essentieel in het begin, maar eens deze ongeveer goed zitten zullen de add en remove zelden nog aanpassingen moeten doen. Dit omdat we meestal rond deze aantallen blijven. Zonder deze regeling

Figuur 4.14: Vergelijk makespan, improving iterations en uitvoeringstijd op X-n139-k19 met uitsluiting van specifieke mutaties, gesorteerd op makespan

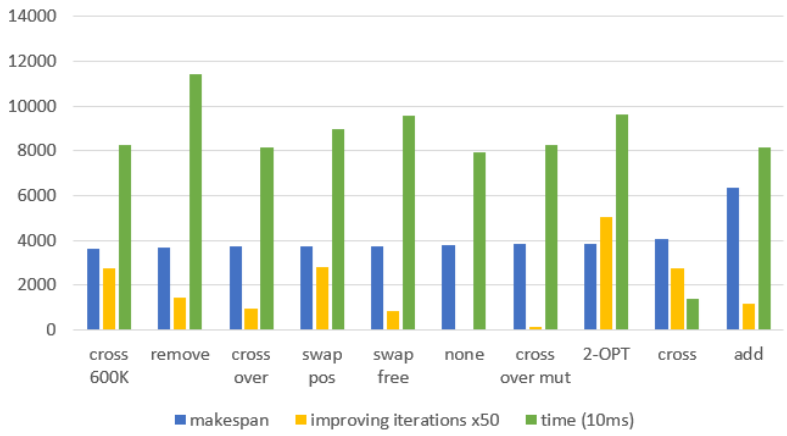


van de aantallen zal de kwaliteit van de oplossingen drastisch slechter worden. Cross-over valt ook buiten de trend, maar dit komt omdat cross-over vaak voor grote veranderingen/verbeteringen zorgen, en dus belangrijk zijn voor het slagen van het genetisch algoritme. De uitvoeringstijden hebben geen opmerkelijke verschillen, we zien enkel dat de cross wat meer van de tijd inneemt. Dit is ook vrij logisch aangezien dit een heuristiek heeft met tijds complexiteit $O(n^2)$ en de andere heuristieken allemaal in lineaire tijd of zelfs constante tijd berekend kunnen worden.

We hebben het bovenstaande experiment ook uitgevoerd op een grotere test instantie, hierbij kijken we vooral naar welke mutaties ook op grote problemen goed blijven presteren en ook naar de uitvoeringstijd en de schaalbaarheid van onze mutaties. We hebben de test op dezelfde manier uitgevoerd, maar deze keer op de test instantie X-n1001-k43 met 22 wagens en 21 drones. De resultaten hiervan staan in figuur 4.15

De verschillen die we hier zien tussen de kleine en de grote instantie zit vooral in de ordening van de beste makespan, we zien nu namelijk sommige mutaties die minder efficiënt worden. Bijvoorbeeld de remove weghalen krijgt zelfs een positief effect. Dit is omdat onze remove heuristiek vooral bestemd is voor kleine instanties. Dit is echter ook instantie specifiek als sommige routes veel compacter kunnen, zullen er meer locaties toegevoegd moeten worden aan routes dan removed in de oplossing. Dus dit kunnen we hier ook zien. Significant is wel dat sommige mutaties die anders wel zeer goed waren zoals swap pos, nu zelfs beter presteren als deze wegvallen. Maar ook dit kan logisch verklaard worden, en ook mutaties zoals cross-over vallen hieronder. Dit zijn mutaties die vaak naar het einde toe nog verbeteringen geven, maar bij 100k iteraties zit de oplossing van een instantie met 1001 nodes nog in de beginfase van het oplossen en een probleem met 139 locaties al in het eindstadium, daarom zien we hier ook de verschillen. Als we kijken naar de uitvoeringstijden valt vooral op hoeveel korter de uitvoering is zonder cross. Zonder cross is het algoritme ongeveer 6 keer sneller voor hetzelfde aantal iteraties. Daarom hebben we ook eens zonder cross gedraaid met 600k iteraties in plaats van 100k zodat we een gelijkaardige tijd krijgen. Hier zien we duidelijk dat dit het beste presteert over een gelijkaardige tijd. Daarom gebruiken we in onze reference oplossing voor grote probleem instanties geen cross mutatie. Een extra optimalisatie kan echter zijn wanneer we een aantal iteratie geen improvement vinden wel terug cross aanleggen zodat het wel gebruikt kan worden om uit lokale minima te geraken.

Figuur 4.15: Vergelijk makespan, improving iterations en uitvoeringstijd op X-n1001-k43 met uitsluiting van specifieke mutaties, gesorteerd op makespan



Hoofdstuk 5

Hybrid genetic search voor drones (HGSD) voor dynamisch PDSVRP

HGSD is ontwikkeld met in het achterhoofd een latere mogelijke uitbreiding voor een online algoritme dat ook voor de dynamische variant van PDSVRP kan oplossen. Er is echter slechts één paper die ook op de dynamische setting van PDSVRP werkt. Namelijk die van Ulmer et al. [62] over same-day delivery. Same-day delivery is dan ook de grootste toepassing van een online algoritme voor PDSVRP. Maar de kans is echter groot dat we met drones op termijn naar een systeem van same-day delivery over kunnen gaan en we dus een dergelijk algoritme zullen moeten gebruiken. Daarom is research naar dynamisch PDSVRP dus uitermate interessant.

5.0.1 Aanpassingen aan HGSD voor dynamisch PDSVRP

De grootste aanpassing die er zal komen bij het dynamisch PDSVRP is dat er nu meerdere routes per shift gereden zullen worden. Hierdoor hebben we nu een extra mogelijkheid bij het inplannen van de routes. We kunnen namelijk een locatie nu ook openlaten voor een volgende set routes. Deze zal in de huidige set routes niet bezocht worden, maar wordt gehouden tot de volgende routes. We hebben nu deze drie statussen waarin een locatie zich kan bevinden:

- **Wagen locatie:** Deze locatie ligt op een route van een wagen en zal de volgende route van de wagen bezocht worden.
- **Drone locatie:** Deze locatie zit in de set van drone locaties en zal door een drone bezocht worden.
- **Skipped locatie:** Deze locatie wordt tijdens de huidige set van routes overgeslagen, maar kan later opgenomen worden in de routes.

De constraint valt hiermee ook weg dat we alle locaties moeten bezoeken aangezien dit nu niet altijd mogelijk zal zijn. Dit omdat we nu ook de time constraint van de shift waarin we werken. Als een locatie toekomt op het einde van de shift zullen we deze bijvoorbeeld niet meer kunnen bezoeken. Daarom kunnen sommige fitness functies zoals totale afstand niet meer gebruikt worden aangezien deze uitgaan van de constraint dat we alles bezoeken.

De manier waarop we onze verdeling maken tussen drones en wagens kan gelukkig vrij makkelijk uitgebreid worden om ook locaties over te slaan. Zo kunnen we in onze set van free locations die normaal allemaal voor drones waren, de meest efficiënte locaties voor drones laten en de rest maken we skipped locaties. Hierdoor is er een beperkte toegevoegde kost voor het bepalen van de efficiëntste drone locaties. Dit komt neer op het sorteren van de locatie volgens toenemende afstand van de depot.

De dichtste locaties zal de drones het efficiëntste kunnen doen. We vullen dus drone locaties in een set tot alle drone routes even lang zijn als de huidige langste wagen route en de rest van de locaties laten we over als skipped locaties. Deze verdeling wordt bijvoorbeeld ook gemaakt in ons dynamisch leveringen/uur algoritme 2.5.4 hierbij plannen we free locations in voor drones tot alle drones werk hebben voor t_{drone} tijd en de overblijvende locaties worden niet in een route opgenomen, en zijn dus skipped locations en worden niet in rekening gebracht in de fitness functie.

Iets anders wat nu moet gebeuren is het kiezen van een volgende route voor wagens. De lengte van de routes voor wagens kunnen namelijk verschillen, en wanneer deze toekomen in de depot moeten we hen een nieuwe route geven. We bespreken twee mogelijke manieren waarop we dit kunnen doen.
De eerste methode gaat bij het kiezen van een route de beste oplossing bevriezen en de overige routes zoals in de gekozen oplossing ook geven aan de andere wagens. Hierdoor zijn we zeker dat de combinatie en verdeling van routes zeer efficiënt is. Na het toewijzen van een oplossing beginnen we gewoon volledig opnieuw voor de volgende set routes met alle free locations wat dus vrij makkelijk te doen is. Wanneer een wagen toekomt kijkt hij eerst of er een bevroren oplossing is, met ongereden routes. Als dit het geval is neemt de koerier de langste ongereden route die hij nog kan rijden. Als er geen ongereden routes meer zijn in de bevroren oplossing nemen we de huidige beste oplossing in de berekeningen en maken deze de nieuwe bevroren oplossing. Maar doordat de routes kunnen verschillen in lengte zal dit zorgen voor ongebruikte rekentijd aangezien de oplossing al bevroren is en er dus geen rekening gehouden is met de locaties toegevoegd tussen het bevriezen van de oplossing en het tijdstip waarop de routes van deze oplossing beginnen.
Een tweede methode die we kunnen doen gaat als volgt: wanneer een wagen toekomt nemen we de beste route van de huidige beste oplossing en verwijderen deze. Hierna blijven we continu verder rekenen aan alle oplossingen ook degene waarvan de route verwijdert is, die route wordt dan terug leeg geïnitialiseerd. Op deze manier zullen we dus altijd blijven doorrekenen en altijd de beste route kunnen geven. Een probleem dat hierbij wel kan ontstaan is dat wanneer wagens kort na elkaar toekomen ze de lading soms ongelijk zullen verdelen. Stel bijvoorbeeld dat er m wagens zijn en er komen twee wagens tegelijk toe dan zal de eerste zijn route ongeveer $r1 = (locs - fls)/m$ van de locatie bevatten waarbij $locs$ alle locaties is en fls alle vrije locaties, maar die van wagen 2 ongeveer $r2 = (locs - fls - r1)/m$ omdat de locaties van $r1$ verwijderd zijn, maar de nieuwe berekende routes opnieuw voor m wagens zullen zijn. Als er dus veel wagens tegelijk binnen komen kan dit een groot probleem worden en zouden we beter het eerste systeem hanteren.

Vaak zullen we ook een systeem hanteren waarbij we willen dat de lengtes van de routes wel ongeveer gelijkaardig zullen zijn. Daarom zal in ons geval met de fitness functie die wij hanteren de eerste keuze ook de beste zijn. Ook is een route verwijderen in de 1e methode bijna geen werk, maar in de 2e methode is dit een pak lastiger.
We kunnen ook als we toegang hebben tot meerdere threads, wel degelijk verder rekenen met de bevroren oplossing door telkens met een route minder verder te rekenen op bevroren oplossing. Hierbij wordt de verwijderde route niet terug leeg geïnitialiseerd maar valt deze route echt volledig weg en rekenen we met een route minder. We beginnen op een andere thread dan tegelijk opnieuw met de free locations uit de bevroren oplossing. Let wel op als een route weggaat van de bevroren oplossing dan moeten nieuwe locaties die na het bevriezen erbij gekomen zijn ook uit de andere populatie met de nieuwe oplossingen verwijdert worden. Er dient dan dus een intersectie genomen worden tussen de verwijderde locaties en de locaties in de nieuwe berekeningen om te zien wel potentieel toegevoegd en verwijdert dienen te worden. Dit is een vrij ingewikkeld en intensief process waardoor we aanraden niet verder te rekenen op de bevroren oplossing en dan gewoon ervoor zorgen dat de routes ongeveer even lang zijn waardoor weinig computation tijd aan de bevroren oplossing verloren gaat.

Bij drones is het makkelijker, deze keren elke keer terug naar de depot waardoor hun volgende locatie wel elke keer aangepast kan worden. Omdat we de efficiëntie zo hoog mogelijk willen maken kunnen

we als volgt te werk gaan. We kunnen telkens de dichtste vrije locatie nemen van de beste huidige oplossing verschillend van de bevroren oplossing. Op deze manier zullen we het meeste leveringen/uur garanderen voor drones.

5.1 dynamic HGSD fitness functie

Doordat we tijdens het berekenen niet alle informatie hebben van het VRP zijn sommige fitness functies niet zo nuttig om te optimaliseren. Denk dan aan fitness functies zoals total disctance of makespan. Het feit dat een koerier meerdere routes kan doen en hierdoor locaties openlaten voor volgende routes zorgt ervoor dat makespan of totale afstand lokaal minimaliseren zal zorgen dat alle locaties skippen voor de beste oplossing zal leiden. Daarom gebruiken we efficiëntie van de routes als een metriek om te optimaliseren. Als het aantal leveringen/uur maximaal is de hele dag door zal het aantal pakjes bezorgt op het einde van de dag ook dicht bij het optimale liggen. Deze metriek zal er ook voor zorgen dat locaties overgelaten kunnen worden zonder problemen namelijk als deze de leveringen/uur zouden verminderen. Er is hier echter wel een klein probleem zoals eerder besproken in algoritme 2.5.4. Wanneer we puur dit optimaliseren zullen korte super efficiënte routes meestal de hoogste score halen en de routes dus telkens klein gehouden worden. Door een initiële startkost p_{load} toe te voegen voor het laden is dit al deels opgelost, maar soms willen we dat de routes nog langer zijn. Want als we steeds korte routes doen zal onze leveringen/uur ook gewoon heel drastisch afnemen over de verschillende sets van routes omdat alle efficiënte kort stukken al gedaan zullen zijn. Daarom voegen we een extra start penalty toe p_{start} wat een soort minimumtijd is dat we willen dat de koerier bezig is. Uiteindelijk maximaliseren we dus de fitness :

$$fitness(P) = fitness_{drones} \sum_{r \epsilon routes} \frac{|r|}{t_r + p_{start}})$$

Hierbij staat $fitness_{drones}$ voor de fitness van de drones berekend volgens het algoritme in 1. t_r staat voor de tijd van de route r inclusief de load en deliver penalty's.

5.2 Dynamische aanpassingen aan PDSVRP

Wanneer we online willen werken op het dynamische PDSVRP zijn er enkele zaken die kunnen gebeuren, er kan een nieuwe locatie bijkomen, een drone kan een trip naar een locatie starten en een route kan gestart worden. Al deze zaken hebben invloed op de volledige populatie, we bespreken hoe we dit moeten aanpakken in HGSD om dit ook te weerspiegelen in de oplossingen van onze populatie.

5.2.1 Locatie toevoegen

Wanneer er een nieuwe levering bijkomt kunnen we deze locatie eenvoudig als skipped locatie toevoegen aan alle oplossingen in de populatie, dit heeft namelijk geen invloed op de fitness waardoor we deze niet moeten herberekenen. Praktisch komt dit neer op het toevoegen van de oplossing aan de free locations van elke oplossing en de fitness niet aanpassen. Hierdoor wordt deze locatie wel opgenomen in de berekeningen en kunnen latere mutaties kunnen deze locatie wel opnemen in een route.

5.2.2 Locatie verwijderen

Wanneer een drone een locatie bezoekt moet deze locatie verwijderd worden uit de oplossing, dit komt neer op het verwijderen van de locatie in alle oplossingen. Als de locatie in de free locations zit moet deze hieruit verwijderd worden, maar als deze in een route zit moet de predecessor van de locatie verbonden worden met de successor. Maar hierdoor verandert de fitness natuurlijk ook van de oplossing dus dienen we het verschil ervan te berekenen met volgende formules.

$$\Delta r = dist(pred(l), succ(l)) - dist(pred(l), l) - dist(succ(l), l)$$

$$\Delta fit = \frac{|r-1|}{(dist_r + \Delta r)/v_t + |r-1| \cdot p_{deliver}} - \frac{|r|}{dist_r/v_t + makespan + |r| \cdot p_{deliver}}$$

Wanneer de locatie van een drone was kunnen we het verschil in fitness schatten door:

$$\Delta fit = \frac{1}{2 \cdot makespan}$$

Wanneer het volstaat een geschatte Δfit voor een wagen te gebruiken kunnen we dezelfde Δfit van drones gebruiken. Maar de formule voor wagens is ook in constante tijd te berekenen dus normaal zou dit geen probleem mogen zijn. Pseudocode van deze operatie is te zien in algoritme 13.

Algorithm 13 removeLocation l in online HGSD

Input: l, *population*

for i in 0..populationSize **do**
  if location in population[i].freeLocations **then**
    $population[i].freeLocations.remove(l)$;
    $population[i].fitness \leftarrow population[i].fitness - 1/population[i].makespan$;
  else
    $r \leftarrow$ route of l;
    $\Delta r = dist(pred(l), succ(l)) - dist(pred(l), l) - dist(succ(l), l)$;
    $\Delta fit = \frac{|r-1|}{(dist_r+\Delta r)/v_t+|r-1|\cdot p_{deliver}} - \frac{|r|}{dist_r/v_t+population[i].makespan+|r|\cdot p_{deliver}}$;
    $population[i].fitness \leftarrow population[i].fitness - \Delta fit$;
  end if
end for

5.2.3 Route verwijderen

Wanneer een route gereden wordt moet deze verwijderd worden, de manier waarop we dit kunnen doen hangt af van de manier waarop we de volgende route kiezen zoals hiervoor aangehaald. Als we geen oplossing bevriezen moeten we alle locaties uit alle routes verwijderen van de populatie. Dit kunnen we doen volgens de methode hierboven aangegeven en erna dienen we ook de fitness aan te passen. Wanneer we echter wel een oplossing bevriezen dienen we gewoon de route te verwijderen uit de bevroren oplossing en aanduiden als gereden, en als er geen route meer is de beste oplossing bevriezen en nieuwe berekeningen starten met de free locations. Wanneer we ook verder rekenen met de bevroren oplossing dienen we bij het verwijderen van een route te kijken of er nieuwe locaties in deze route zaten zo ja moeten deze ook verwijderd worden uit de rest van de nieuwe populaties. En dient bij het verwijderen van de laatste routes gekeken te worden bij de free locations of er geen tussen zitten die nog niet in de nieuwe populatie zit doordat deze verandert kan zijn tijdens het berekenen. Pseudocode van deze operatie is te vinden in algoritme 14.

5.3 Eindigen van HGSD in dynamische PDSVRP

Een ander probleem dat zich voordoet bij het dynamische PDSVRP, is hoe kunnen we op de beste manier eindigen. Wanneer we weten dat onze shift ten einde loopt dan hebben we het probleem waarbij sommige koeriers hun laatste routes rijden, we willen dan dat nieuwe routes nog zoveel mogelijk locaties meenemen en niet zoals gewoonlijk de locaties blijven verdelen onder de m wagens.
Wat we hieraan kunnen doen is de laatste berekeningen doen voor het aantal koeriers dat nog niet aan hun laatste route bezig is. We kunnen namelijk bij het bevriezen van de oplossing perfect weten hoelang elke route gaat duren en welke koeriers dus nog een nieuwe route zullen kunnen starten. Als

Algorithm 14 getRoute in online HGSD

Input: *population*, $frozenSol$

if $frozenSol.routes.size() = 0$ **then**
  $frozenSol \leftarrow population[0]$;
  run HGSD with free locations in $population[0]$;
end if
$sol \leftarrow$ longest route in $frozenSol$;
remove sol from $frozenSol.routes$;

return sol;

we aan het einde van de shift zitten kunnen we dan de nieuwe berekeningen starten voor het aantal routes dat nog gereden wordt. Op deze manier kunnen we een clean einde hebben.

Hoofdstuk 6

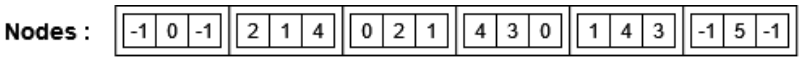
Implementatie PDSVRP

We hebben ook een reference implementatie gemaakt van PDSVRP die de ideeën uitgewerkt in deze paper implementeert, deze is te vinden op Github. Daarbij hebben we vaak inspiratie gehaald bij de HGS implementatie van vidal et al [66].
Voor het implementeren van VRP, HGS en PDSVRP zijn er enkele zaken waarmee we rekening moeten houden, en een hoop optimalisaties die we kunnen maken. Deze zaken bespreken we in dit hoofdstuk, samen met een algemene aanpak van implementatie van sommige concepten.

Implementatie is een vrij belangrijk onderdeel van HGS aangezien het slagen van het algoritme afhankelijk is van de snelle uitvoering van de mutaties. Wanneer we dus inefficiënties hebben in de implementatie of net optimalisaties hebben, zal dit ook een groot verschil kunnen geven aan de performance en resultaten van ons algoritme. Niet alle besproken optimalisaties uit de paper zijn ook in de reference code geraakt, maar ze worden hier toch gegeven omdat ze nuttig zijn voor mogelijks verder onderzoek. Of een volgende versie van het algoritme.

6.1 Oplossing voorstelling

Allereerst was het belangrijk een goede voorstelling te vinden voor het probleem, hierbij is het belangrijk dat deze, compact is, makkelijk te manipuleren en makkelijk te clonen. Dit laatste is belangrijk omdat wanneer we een mutatie doen van een oplossing we deze telkens dienen te clonen omdat onze mutatie in-place gebeurt en we ons origineel niet mogen verliezen. Veel van onze mutaties zoals cross, cross-over, ... zijn ook mutaties die kleine veranderingen doen op basis van de connecties in de route, maar als we kijken naar de verandering op de totale array is dit wel vrij significant. Bij een cross bijvoorbeeld verwisselen we eigenlijk slechts 2 connecties in 2 routes, maar hierdoor veranderen grote delen van deze routes. Als we onze routes zouden voorstellen als arrays is dit een grote kost voor een kleine aanpassing. Ook bij het toevoegen en verwijderen van locaties in een route verandert de lengte van de array, wat zware operaties kunnen zijn voor arrays als deze geen vrij geheugen meer hebben en dus de memory verplaatst dient te worden. Om deze redenen gebruiken we gelinkte lijsten. Deze zitten ook op een gelijkaardige manier in elkaar als onze routes, en hier kunnen we een operatie zoals een cross doen door 2 pointers aan te passen, wat dus een pak efficiënter is. En ook locaties invoegen of verwijderen gaat hier ook een stuk sneller.
Het grote probleem met linked lists is echter het clonen. Dit omdat het geheugen niet achter elkaar dient te zitten en we telkens elk element van de list één voor één moeten over kopiëren. Wat we gedaan hebben om dit probleem te verhelpen is gelijkaardig aan de aanpak van Vidal, maar met een extra twist. We weten dat elke locatie slechts eenmaal in een route of vrije locatie mag zitten. We kunnen dus voor elke locatie een unieke node maken die de *id* van de node bevat en de predecessor en de successor. De ids zijn de indices van de locaties en gaan dus van 0 tot het totaal aantal locaties. We



Figuur 6.1: Voorbeeld van nodes representatie van de route [0,2,1,4,3,0] and free location 5.

maken een array van deze nodes waar ze als index in de array hun id hebben, hierdoor kunnen we direct een look-up doen van location id naar node in de linked list. Wanneer we nu een linked list willen over kopiëren kunnen we alle elementen in één keer over kopiëren door de array te kopiëren en erna de pointers terug juist te zetten, maar ook dit kan makkelijk door de memory offset bij alle pointers op te tellen. We kunnen het aanpassen van de pointers zelfs verder vereenvoudigen door in plaats van met pointers te werken, met de indices te werken in de array. Zo is de linked list meer human-readable en kunnen we de linked lists in zijn geheel over kopiëren zonder enige aanpassingen te hoeven doen. We hebben dus in elke nieuwe node de index van de successor, de huidige location id en de index van de predecessor. Een representatie hiervan is te vinden op figuur 6.1. Verder wordt een locatie ook overal in het algoritme voorgesteld als zijn id. Om afstanden te bepalen hebben we initieel een afstandsmatrix opgesteld die de afstanden tussen de locaties opslaat, op deze manier berekenen we ook slechts de afstanden eenmalig.

We hebben naast nodes in onze representatie nog een lijst van $freeLocations$ die aangeeft welke locaties vrij zijn en ingepland dienen worden door drones. In de nodes representatie worden deze aangegeven door hun successor en predecessor op -1 te zetten. Bij 0 doen we dit ook omdat 0 de depot voorstelt en deze heeft meerdere successors en predecessors door alle routes die hieruit vertrekken. We slaan ook nog een lijst van routes op die voor elke route de start index, lengte van de route en fitness van de route bijhoudt.

6.1.1 implementatie biased fitness functie

De fitness bepalen van een route is een kritiek deel van ons algoritme en hier kruipt ook een groot deel van de resources in. Dit omdat na elke mutatie die we doen dient gekeken te worden of we een verbetering in de fitness functie hebben kunnen verwezenlijken. De biased fitness functie bestaat uit de afstand van de routes te bepalen, de diversiteit contributie ΔP te bepalen, de drone planning gretig uit te voeren en de formule hierop toe te passen (makespan, leveringen per uur, ...). De formule zelf is meestal geen zware kost de kritieke delen zijn het bepalen van de afstanden van de routes en de ΔP.

Bij het bepalen van de afstanden is er een eerste optimalisatie die we kunnen doorvoeren door de fitness van de routes bij te houden en enkel de fitness van aangepaste routes te herrekenen. Op deze manier zullen we nooit de fitness van eenzelfde route tweemaal berekenen. We kunnen bij de meeste mutaties ook snel een verschil in fitness of $\Delta fitness$ bepalen van een route. Wanneer er bijvoorbeeld een locatie toegevoegd wordt kunnen we gewoon het verschil berekenen en toevoegen aan de huidige fitness functie van de route.

Zoals besproken in sectie 4.4.3 kunnen we enkele optimalisaties ook doen aan het berekenen van de diversification contribution. Deze zijn vooral het minimaliseren van het aantal keren dat we de diversiteit moeten berekenen tussen routes en het gebruiken van dynamisch programmeren voor het berekenen van ΔP. We kunnen op een gelijkaardige manier als bij de fitness functie hier ook verder in gaan en het verschil in diversiteit door een mutatie berekenen. We kunnen dit doen aan de hand van algoritme 15. *parent* is de oplossing waarop de mutation gedaan is en *solution* is de oplossing waarvan we de diversiteit willen bepalen. Bij het vergelijken van de successors en de predecessors is de uitkomst één als deze *true* is en 0 als ze $false$ is. Door deze optimalisaties kunnen we het berekenen van de diversiteit reduceren van het vergelijken van alle successors en predecessors van een route naar het vergelijken van enkel degene die veranderd zijn.

Algorithm 15 Bereken verandering in diversity

Input: *mutation*, *parent*, *solution*

diversity ← diversity of parent;
old ← 0;
new ← 0;

for *node* ∈ nodes with changed pointers **do**
old ← *old* + (*parent.nodes*[*node*].*succ* ≠ *solution.nodes*[*node*].*succ*) + (*parent.nodes*[*node*].*pred* ≠ *solution.nodes*[*node*].*pred*);
new ← *new* + (*mutation.nodes*[*node*].*succ* ≠ *solution.nodes*[*node*].*succ*) + (*mutation.nodes*[*node*].*pred* ≠ *solution.nodes*[*node*].*pred*);
end for

diversity ← *diversity* − *old* + *new*;

return *diversity*;

Bij het inplannen van de drones is er niet echt een grote optimalisatie die gedaan kan worden. Wanneer er geen aanpassing is aan de vrije locaties zoals bij een cross of een cross-over mutatie dient de drone planning niet herrekend te worden. Maar anders berekenen we deze telkens met een longest-proces-first heuristiek. Deze heeft een tijdscomplexiteit van $O(n\,log(n))$. Een mogelijke optimalisatie kan zijn om dit in het begin benadert te doen $(4/3 + 1/3m) \cdot dist_{drones}/drones$. Dit is een schatting van de drone kost, maar deze is echter niet exact, dus als we later de exacte kost willen hebben dienen we wel telkens het inplannen van de drones te doen. Daarom kunnen we naar het einde toe misschien toch terug overschakelen naar een exacte fitness.

6.2 Mutaties optimaliseren

Voor vele mutaties bestaan er specifieke methodes waarop we deze kunnen optimaliseren. Het algemene idee van veel optimalisaties is het gebruiken van de lokaliteit van nodes om te bepalen of verbeteringen waarschijnlijk zullen zijn. Daarom hebben we dus vaak heuristieken die de neighbors van een locatie opvragen. Om dit op een efficiëntie manier te doen hebben we de coordinate-space opgedeeld in een grid en per tegel van het grid een bin gemaakt met een set van alle locaties in die tegel, als we de buren willen van een locatie kunnen we de bin van zijn tegel geven en mogelijks de bins van omringende tegels als ze verder willen zoeken. Op deze manier kunnen we in door preprocessing van de locaties de neighbors later opvragen in $O(1)$ tijd.

Een andere optimalisatie die uitgevoerd is, is het in-place muteren van mutaties. Door onze efficiënte representatie kunnen we met een memory copy een oplossing efficiënt clonen en hier de mutatie in-place uitvoeren op de clone. Hierdoor moeten we enkel initieel memory alloceren voor alle oplossingen in een populatie en hergebruiken we hierna telkens de memory van oplossingen die de generatie niet overleefd hebben.

Doordat we met een gesorteerde dataset werken voor het bijhouden van onze populatie en het berekenen van de biased fitness vooral afhankelijk is van onze survivors kunnen we onze populatie inkrimpen tot het aantal survivors +1. Onze populatie is een ordered set die bestaat uit een paar van de berekende biased fitness en de oplossing. Als we een nieuwe mutatie hebben voegen we deze toe aan de ordered set en de volgende mutatie wordt gedaan in-place op het nieuwe paar met laagste score. En zo kunnen we iteratief verder zoeken. Het verkleinen van de populatie heeft een verlaging op de

memory footprint, maar is vooral ook een optimalisatie voor onze diversification functie, deze dient namelijk van elke oplossing van de populatie een diversificatie score mee te geven, dus hoe kleiner de populatie hoe efficiënter we deze kunnen berekenen.

6.3 parallelliseren

Een groot voordeel van het gebruiken van random restarts is dat dit zeer gemakkelijk te parallelliseren valt. We kunnen elke random restart op een aparte thread draaien en deze hebben geen communicatie tussen elkaar nodig. Zo kunnen we makkelijk worker threads maken die enkel nieuwe random restarts maken en draaien, en andere processor threads die deze random restarts gebruiken om de hiërarchie op te vullen en verder te zoeken in een populatie. Zeker bij onze ketting-hiërarchische structuur is dit zeer gemakkelijk aangezien we altijd random restarts nodig zullen hebben elk level in de hiërarchie, maar de oplossingen, doordat ze random restarts zijn, niet afhankelijk zijn van elkaar zoals in de normale hiërarchie. We kunnen in dit geval over gaan naar een parallel producer/consumer patroon waarbij enkele threads met random restarts nieuwe oplossing produceren en andere threads deze oplossingen consumeren en nieuwe populaties maken of deze invoegen in hun populatie om een nieuw level te beginnen in de hiërarchie.

Hoe we dit concreet opgelost hebben in onze oplossing is met behulp van één hoofd thread en de rest worker threads. De hoofd thread zijn taak is telkens het hoogste level in de hiërarchie te trainen en hierna een nieuw hoger level te starten en op te vullen met de beste oplossingen van de random restarts. De worker threads vullen een gesorteerde set van oplossingen van random restarts aan. Wanneer deze te lang wordt start een worker zelf een populatie met oplossingen van de random restarts en voegt het resultaat hiervan terug toe aan de lijst van random restarts. Dit komt eigenlijk neer op een oplossing die een extra level hiërarchie heeft doorgemaakt. Dit heeft dus als extra voordeel dat als we met veel worker threads werken er ook veel oplossingen zullen toegevoegd worden aan de hoofd thread die meerdere levels van hiërarchie hebben meegemaakt en dus van hogere kwaliteit zullen zijn.

Hoofdstuk 7

Experimenten en resultaten

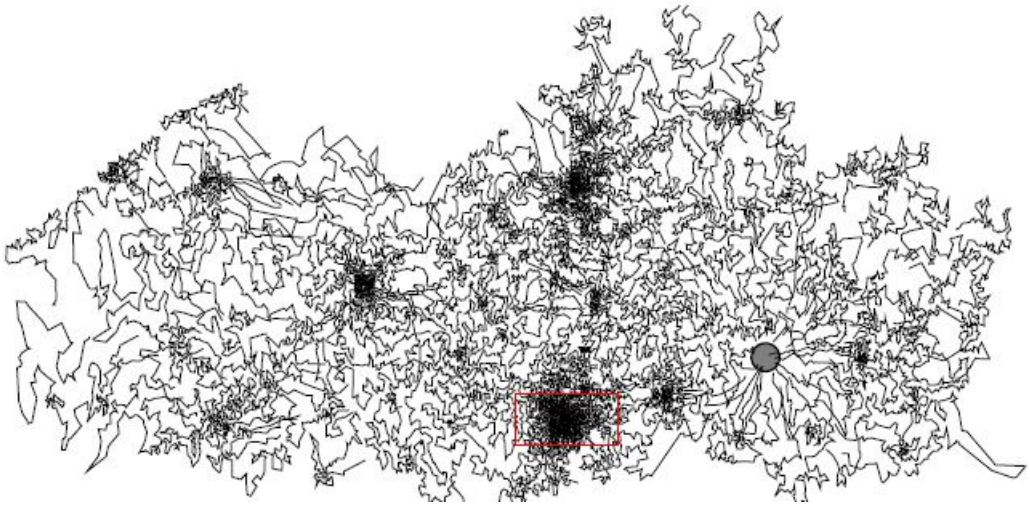
Alle testen en experimenten die we uitgevoerd hebben zijn gebeurd op de HPC van de Universiteit Gent met volgende specificaties 2 x 18-core Intel Xeon Gold 6140 (Skylake @ 2.3 GHz), 88 GiB RAM (4GiB allocated).

7.1 Benchmarks & realistische scenarios

We willen natuurlijk ook weten hoe goed onze oplossing presteert ten opzichte van andere oplossingen, en hoe dicht we zitten bij de huidige state-of-the-art. Om dit op een objectieve manier te kunnen meten gebruiken we standaard benchmarks van VRPLIB [71], deze bevat veelgebruikte VRP benchmarks zoals die van Golden et al. [25] en Uchoa et al. [61] maar ook tal van andere. We gebruiken ook enkele instanties van TSPLIB [56] zodat we ook kunnen vergelijken met PDSTSP oplossingen aangezien er nog niet veel PDSVRP oplossingen beschikbaar zijn. We nemen deze benchmarks aangezien verscheidene papers al met deze datasets werken en hierdoor kunnen we makkelijk meten hoe we performen ten opzichte van de anderen. Zo maken we het ook mogelijk voor later onderzoek om makkelijk te testen en vergelijken met onze oplossingen. De best known solutions die meegegeven worden bij TSPLIB en VRPLIB zijn echter wel oplossingen voor cVRP en klassiek TSP. Toch valt hier nog steeds goede info uit te halen omdat aangezien we ons algoritme ook zonder drones kunnen laten draaien of enkele koeriers kunnen vervangen door drones en zo de impact van het introduceren van drones meten. We bespreken hierna ook de manieren gebruikt in de literatuur om deze problemen om te vormen naar PDSVRP en PDSTSP problemen, en daarna ook een eigen manier waarop we voorstellen dit in de toekomst te doen.

Een van de doelstellingen van deze paper was ook om realistische scenario's te bekijken en het ook mogelijk te maken competitieve oplossingen voor large-scale PDSVRP instanties te produceren. Er is namelijk nog geen paper die resultaten voor deze large-scale VRPs uitgebracht heeft. Daarom hebben we ook gezocht naar large-scale en realistische benchmarks en scenario's om tegen te testen. De benchmark set die we hiervoor gevonden hebben is deze uit Arnold et al. [4]. Deze heeft een benchmark gedaan voor realistische leverscenario's in Vlaamse steden die een realistische afstand en aantal locaties hebben (figuur 7.1). Ook hebben ze in dezelfde paper veel realistische data gevonden over de capaciteit van koerierwagens en dergelijke die we ook proberen gebruiken. Hun instanties zijn ook terug te vinden in CVRPLIB [71].

We hebben deze manier van werken gekozen aangezien vaak in papers over large-scale VRPs [4] aangehaald wordt hoe de probleem groottes van huidige datasets vaak afgeleid zijn van de eerste datasets zoals Golden et al. (1998) [25] en hierop verder gebouwd zijn. Aangezien de meeste VRP algoritmes nu ook voor deze probleem grootte gemaakt en geoptimaliseerd zijn is het soms moeilijk om de switch te maken naar grote datasets met veel leveringen per koerier. Vaak zijn de grootste VRP datasets slechts tot 250 locaties groot en hebben deze dan al een groot aantal koeriers. Hetgeen wat het



Figuur 7.1: Oplossing uit de paper van Arnold et al. op hun realistische VRP dataset in Vlaanderen

minst overeenkomt met de realiteit is hier vooral het aantal locaties per koerier, wat in veel datasets slechts 10-20 locaties per koerier zal hebben. Zoals aangehaald in de paper van Arnold et al. [4] zijn realistische routes van koeriers een stuk langer dan de lengte van routes in huidige datasets. Volgens hen worden routes ook net langer en zijn routes uit 1998 niet meer te vergelijken met hedendaagse routes voor leverdiensten. In een bestelwagen kunnen tegenwoordig 100-200 pakketten meegenomen, met een mild geschatte levertijd van rond de 5 minuten per pakket en 8 werkuren per dag kan een route dus een 100 tal locaties bezoeken op een dag wat toch een grootte orde meer is dan de 10-20 pakjes in veel datasets. Golden et al. [25] heeft dan als een van de weinige nog instanties tot 420 locaties en ook instanties met 48 locaties/koerier. De instantie waarmee het meeste getest wordt is Uchoa et al. [61] deze ook meestal 10-20 pakjes/koerier maar heeft wel instanties tot 1000 locaties. De instanties van Li et al. zijn de enige die ook 50-100 pakjes/ koerier hebben, maar deze zijn geïnspireerd op Golden_4 en zijn volgens een patroon en niet volledig random wat ze ook minder bruikbaar maakt voor ons geval. Hierdoor zijn de instanties van Arnold et al. [4] de enige huidige instanties die een realistisch aantal locaties en locaties/koerier heeft als we kijken naar grote koerierdiensten.

7.2 Experimenten in de literatuur

Om een degelijke methodiek te vinden voor het opstellen van onze tests en experimenten gingen we op zoek in de literatuur naar standaard testmethodes voor PDSTSP en PDSVRP. Doordat deze tak van research van VRP nog in zijn kinderschoenen staat zien we echter veel verdeeldheid over de manier van testen, wat het onderzoek niet ten goede komt. Hierdoor zijn er reeds veel verschillende manieren om de oplossing van een PDSTSP te evalueren. Allereerst zien we een opsplitsing tussen de verschillende objectieven, of hetgeen waarnaar we optimaliseren. De meeste papers optimaliseren de makespan of de operationele kosten. Wij hebben voor de makespan gekozen, maar zelfs bij de makespan is er verdeeldheid over de manier waarop we deze berekenen, en vooral hoe we het verschil tussen drones en wagens uitdrukken. In de originele paper van Murray et al. [48] werd gekozen om het verschil aan te geven door een verschil in snelheid en een andere manier van afstand berekening, zo werd de afstand voor wagens berekend met de Manhattan distance en de afstand voor drones met Euclidische afstand. Voor de snelheid werd getest op verschillende configuraties van snelheden van drones en wagens. De makespan werd gegeven in aantal uren werk tot volledige voltooiing. De locaties werden geplaatst in een omgeving van 8 mijl op 8 mijl met verschillende posities van de depot. Er werd hier ook een maximaal uithoudingsvermogen (in tijd) gegeven aan de drones en een percentage ontoegankelijke

locaties voor drones. De experimenten waren random gegenereerd en daarom niet objectief na te bootsen met exact dezelfde locaties. In papers die volgen zagen we gelijkaardige experimenten die deze van Murray et al. probeerden na te bootsen of eigen random testen deden. Tot de paper van Salue et al. (2018) [46] die een nieuwe testmethode voor PDSTSP had ontwikkeld die wel makkelijker na te bootsen valt. Hierbij wordt getest op instanties van TSPLIB die aangepast zijn om te dienen als instanties voor PDSTSP. De aanpassingen die gedaan werden waren de volgende:

- Voor de depot werd een extra locatie toegevoegd in het centrum van de locaties.
- De drone speed werd aangepast met behulp van een drone speed factor die aangeeft hoeveel sneller een drone is dan een wagen
- Er werd een aantal drones meegegeven
- De drone ontoegankelijke locaties werden als volgt gekozen: als we K% drone toegankelijke locaties willen hebben wordt eerst elke locatie met index $n \equiv 0 \mod \lfloor \frac{1}{(1-K)\cdot 0,3} \rfloor$ op ontoegankelijk gezet, hierna werden de locaties gesorteerd volgens toenemende afstand tot de depot en worden deze één voor één aangewezen aan de drone-toegankelijke locaties tot K% drone toegankelijk is, de overblijvende verste locaties van de depot blijven ontoegankelijk voor drones.

Deze manier werd erna door verschillende papers gebruikt om hiermee de methodes objectief te kunnen vergelijken door onder andere Dell'Amico et al [17] en Raj et al. [55]. Bij Raj was er echter een subtiel verschil, hier werden de ontoegankelijke locaties random gekozen in plaats van deze volgens de methode van Salue et al. te nemen. Later in 2022 bracht Salue et al een nieuwe paper uit over PDSmTSP, een generalisatie met m wagens in plaats van één, hier worden test instanties van CVRPLIB gebruikt deze keer. Deze worden op een eenvoudigere manier aangepast als volgt:

- Alle locaties worden op drone toegankelijk gezet.
- Het aantal wagens en drones wordt gezet met volgende formules: $Drones = \lfloor \frac{fleetSize}{2} \rfloor$ en $Trucks = \lceil \frac{fleetSize}{2} \rceil$.
- Om het verschil tussen drones en wagens aan te geven wordt de afstand voor wagens berekend met Manhattan afstand en voor drones met Euclidische afstand, maar dit keer is er geen factor om aan te geven hoeveel een drone sneller gaat dan een wagen.

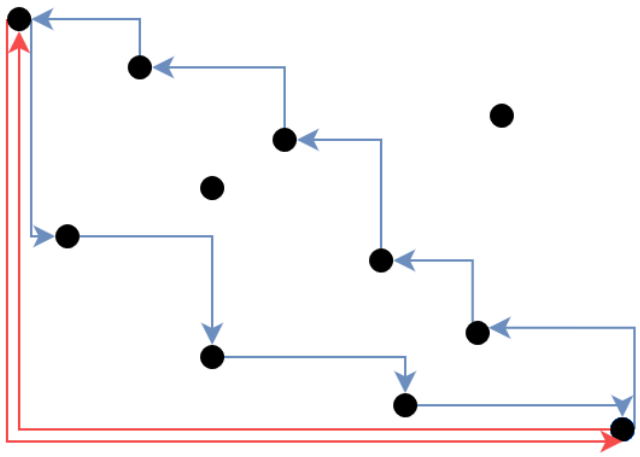
7.3 Eigen aanpak PDSVRP experimenten

Door deze verschillende methoden in het voeren van experimenten van PDSTSP en PDSVRP is er nood aan een logische standaardmanier voor het evalueren van de problemen, waardoor het makkelijk wordt de algoritmes te vergelijken. Daarom gebruiken wij voor onze oplossingen een manier die objectief is en gedetailleerd beschreven staat en makkelijk uitgevoerd kan worden door andere onderzoekers. Dit met het in acht nemen van de verschillende deelproblemen, varianten en implementaties. In de hoop dat in de toekomst makkelijk vergeleken kan worden met deze oplossingen en dat de methode verder gehanteerd wordt. De huidige experimenten in de literatuur hebben ook significante nadelen die nog niet besproken zijn, maar in ons onderzoek naar boven gekomen zijn waardoor een nieuwe standaardmethode gemaakt is die deze problemen probeert te verhelpen.

Op het moment van schrijven van deze scriptie is er geen standaard dataset beschikbaar, maar dit is geen probleem aangezien de standaard cVRP datasets voor onze doeleinden ook gebruikt kunnen worden. We gebruiken datasets van CVRPLIB [71]. Deze bevatten locaties die zowel voor een VRP als een TSP gebruikt kunnen worden, en hebben capaciteiten die gebruikt kunnen worden voor het bepalen van drone-toegankelijke klanten, of later gebruikt kunnen worden voor varianten op PDSVRP die ook rekening houden met de capaciteiten van locaties. Hier dient ook een maximumcapaciteit van

een drone gedefinieerd te worden waaruit ook direct volgt welke locaties wel en niet door drones gedaan kunnen worden. In bijna alle huidige papers wordt een percentage van de locaties als onbeschikbaar voor drones gezet [48] [17] [46], in papers zoals Murray et al. [48] wordt dit random gekozen, een percentage van de locaties zijn random geselecteerd om onbereikbaar te zijn voor drones, hierover hebben we al gezegd dat dit niet objectief na te bootsen valt. Bij de methode van Salue et al [46], die we hiervoor besproken hebben, zijn er ook enkele bedenkelijke keuzes gemaakt die niet zo logisch waren naar ons gevoel. Het getal 0,3 wordt als constante meegegeven en heeft weinig redenering en ook valt er iets te zeggen over verre locaties ontoegankelijk maken, aangezien deze zogezegd aan de rand van de stad liggen en niet in de range zijn van een drone. Maar dit is niet echt een objectieve metriek en is op zich ook een heuristiek aangezien verre locaties door drones laten doen meestal tot slechtere oplossingen leidt. Het toevoegen van deze heuristiek kan leiden tot oplossingsmethoden die hierop inspelen wat geen wenselijke eigenschap is voor een testmethode. Een objectieve manier die we kunnen gebruiken voor het selecteren van ontoegankelijke locaties is om dit te doen aan de hand van de capaciteiten die reeds bestaan in CVRPLIB instanties. We kunnen op twee manieren te werk gaan, we kunnen een maximumcapaciteit op een drone zetten, dan zijn alle ontoegankelijke locaties degene met een hogere capaciteit, ofwel kunnen we te werk gaan met een percentage toegankelijke locaties K zoals in de meeste papers. Hierna nemen we de $\lfloor n \cdot (1 - K) \rfloor$ locaties met de hoogste capaciteit en maken deze ontoegankelijk voor drones. Hierbij sorteren we eerst volgens afnemende capaciteit en daarna volgens index. Dit lijkt een goede aanpak die exact reproduceerbaar is. Het kiezen van een maximum drone capaciteit lijkt de keuze die de realiteit het meest weerspiegelt, maar let hierbij wel op dat de cVRP instanties hier niet op aangepast zijn, en er waarschijnlijk voor verschillende instanties ook een verschillende maximum drone capacity gekozen zal moeten worden. Door de capaciteiten van test instanties aan te passen kan ook onze testmethode ook naar andere testmethodes omgevormd worden.

Een andere belangrijke zaak die gekozen moet worden bij het evalueren van PDSVRP is het verschil tussen drones en wagens. Wanneer we in de literatuur kijken zien we meestal dat de afstand voor drones de Euclidische afstand is en voor wagens de Manhattan afstand, dit is naar het voorbeeld van de papers van Murray et al. [48]. Het idee hierachter is dat er een verschil bestaat in afgelegde afstand tussen een drone en een wagen aangezien een drone niet gebonden is aan het wegennetwerk en in volgelvlucht kan vliegen. Maar dit heeft ook een aantal kritische nadelen die het naar mijn mening onbruikbaar maken. Door Manhattan distance te gebruiken valt onder andere moeilijker te vergelijken met andere VRP varianten aangezien de afstanden verschillen. Hierdoor valt het bijvoorbeeld moeilijker in te schatten wat de winst of verlies is bij het inzetten van drones en kan dit niet rechtstreeks vergeleken worden met de cVRP oplossingen aangezien deze Euclidische afstand gebruiken voor wagens. Ook heeft de Manhattan afstand enkele ongewenste eigenschappen: zo zullen locaties die dichter bij de diagonalen van de depot liggen een groter verschil in afstand hebben bij Manhattan vs Euclidisch en de afstanden horizontaal of verticaal van de depot zullen bijna geen verschil hebben. In realiteit is dit natuurlijk ook soms het geval dat sommige locaties meer winst hebben in afstand door een drone. Maar het grootste bezwaar tegen Manhattan distance is het feit dat deze bepaalde eigenschappen heeft die het bekomen van goede logische oplossingen in de weg kan staan. Bij Manhattan distance kan je namelijk tussen 2 punten meerdere paden maken met dezelfde lengte zolang de paden maar altijd in dezelfde richting gaan als het punt. Dit is een zeer slechte eigenschap voor een VRP. Zo hebben routes die een verre locatie diagonaal moeten bezoeken een even grote afstand als een route die arbitrair veel locaties tussen de depot en de verre locatie bezoekt zolang we enkel de richtingen uitgaan van de verre locatie. Dit wordt geïllustreerd op figuur 7.2. Hier is de afstand en dus de makespan van de ene route gelijk aan de andere. Hierdoor is vaak bij grote instanties met veel koeriers de ondergrens van de makespan van de routes met Manhattan distance gelijk aan de Manhattan distance naar het meest uiterste punt diagonaal van de depot. En dit is toch een zeer ongewenste eigenschap, waardoor het gebruik van Manhattan niet meer logisch te verklaren valt. Hierdoor zijn sommige oplossingen ook mogelijks slecht aangezien de makespan hier serieus kan verschillen van de gemiddelde makespan van goede routes, en hierdoor hebben goede oplossingen even grote makespan als slechte oplossingen zolang hun route naar



Figuur 7.2: Het grote probleem met Manhattan afstand, de blauwe route is even lang als de rode route

de verste node maar efficiënt is. Door Euclidische afstand valt dit grotendeels weg omdat de makespan dan vaak dichter bij elkaar zal kunnen liggen, zo zal een rechtstreekse route enkel even lang zijn als de locaties op dezelfde rechte liggen waardoor lange routes korter gemaakt kunnen worden door locaties er van tussen te halen. Wanneer we makespan gebruiken is het natuurlijk altijd zo dat routes met een lagere afstand dan de langste route altijd slechter kunnen gemaakt worden zonder de totale makespan aan te passen zolang ze niet langer worden dan de langste route.

Een metriek die echter iets algemener is en vaak in combinatie gebruikt wordt met een verschil in afstand is een snelheidfactor voor drones. Deze toont aan hoeveel keer sneller of trager een drone is dan een wagen. Dit werd al geïntroduceerd in Salue et al. [46] in combinatie met Manhattan distance. Vaak wordt dit ook gedaan aan de hand van het meegeven van een snelheid voor een drone en wagen zoals in Murray et al. [48]. Dit is echter equivalent aan het geven van een snelheidsfactor aan een drone f_d.

$$\frac{t_d}{v_d} > \frac{t_t}{v_t} \Leftrightarrow t_d \cdot \frac{v_t}{v_d} > t_t$$

Hierbij is t_t gelijk aan de lengte van een route van een wagen en t_d gelijk aan de lengte van een route van een drone. We zien dus dat we een snelheid constante f_d kunnen gebruiken gelijk aan $\frac{v_t}{v_d}$ voor het berekenen van de makespan. In deze constante kunnen we ook de geschatte winst voor het gebruiken van vogelvlucht tegenover het wegennet schatten en verwerken. Het grote voordeel hiervan is dat de afstand van wagens ongewijzigd blijft, hierdoor kunnen we rechtstreeks vergelijken met VRPs met wagens die de Euclidische afstand gebruiken, zoals de meeste VRPs. Onze factor f_d is dus een combinatie van snelheid van drones versus wagens en winst vogelvlucht versus wegennet. Als we kijken naar het verschil van Euclidische afstand tegenover Manhattan afstand kunnen we dit schatten door het gemiddelde te nemen van de Manhattan distance van punten op de unit cirkel. Wanneer we dit berekenen krijgen we $diff = \frac{4}{\pi} \approx 1,27$. Wanneer we het verschil van snelheid willen bekijken zien we in de literatuur vaak dat dit verschil vrij klein wordt geschat/gehouden zoals bijvoorbeeld in Nguyen et al. [50] en Murray et al. [48] waarbij wagen snelheid rond de 30 km/h genomen wordt en drone snelheid 40 km/h. Wanneer we echter kijken naar de huidige prototypes van drones door Amazon [3] en Wing [70] worden deze geadverteerd met 80 km/h - 104 km/h. Wat dus een veel groter verschil is. Hierdoor wordt onze factor voor snelheid alleen al $\frac{90km/h}{30km/h} \approx 3$. Een factor 3 lijkt dus zeker een correcte waarde voor onze snelheidfactor voor drones f_d. We zien ook dat in de enkele papers al de snelheidfactor 2 in combinatie met Manhattan gebruikt wordt zoals in Salue et al. [46], 3 als combinatie van beide lijkt dan ook een eerlijke waarde.

We gebruiken als depot dezelfde als van het cVRP. Andere belangrijke constraint die we niet toepassen is dat we geen rekening houden met de range van een drone, we gaan er dus vanuit dat alle locaties altijd binnen de range van de drone vallen. Ook hier kunnen algoritmes die hier wel rekening mee houden makkelijk aangepast worden door de range uit te breiden tot het volledige gebied.

Een totale pseudocode voor het berekenen van de makespan wordt gegeven in algoritme 16.

Algorithm 16 makespan

```
makespan ← Ø;

for r ∈ routes do
  if dist_r > makespan then
    makespan ← dist_r;
  end if
end for

for d ∈ drones do
  if dist_d/cr_d > makespan then
    makespan ← dist_d/f_d;
  end if
end for

return makespan;
```

Hierbij zijn $dist_r$ en $dist_d$ de Euclidische afstand van de routes van wagens en drones.

Deze vorm kan ook makkelijk uitgebreid worden om ook penalty's voor bijvoorbeeld laden en lossen te werken, we gebruiken effectief de tijd van een route als makespan in plaats van de afstand. We berekenen de tijd voor een wagen als:

$$t_r = dist_r/v_t + |r| \cdot p_{deliver} + p_{load}$$

en voor drones:

$$t_d = dist_d/v_d + |d| \cdot p_{deliver} + |d| \cdot p_{load}$$

Hierbij is p_{load} en $p_{deliver}$ de tijd penalty's voor laden en lossen voor een drone, en v_t en v_d de snelheid van wagens en drones, deze keer wel in km/h. In algoritme 16 dient $dist_r$ vervangen te worden door t_r en $dist_d \cdot f_d$ vervangen te worden door t_d.

Als samenvatting geven we nog deze opsomming van onze testmethode:

- Kies een cVRP test instantie (uit CVRPLIB)
- Niet drone toegankelijke locaties: voor $K\%$ drone toegankelijke locaties neem de $\lfloor n \cdot (K-1) \rfloor$ locaties met de hoogste capaciteit en laagste index
- Neem de depot van het cVRP probleem
- Minimaliseer makespan volgens afstand met een reducerende snelheidfactor voor drones zoals in algoritme 16
- Alle locaties vallen binnen de drone range
- Capaciteit van wagens wordt niet in rekening gehouden
- Gebruik zowel voor drones als wagens de Euclidische afstand tussen de punten

Tabel 7.1: Variabelen die gebruikt zijn voor het testen

Variable	default value	description
P	10	population size
V	$\lceil \frac{fleetSize}{2} \rceil$	amount of trucks
D	$\lfloor \frac{fleetSize}{2} \rfloor$	amount of drones
K	80%	Percentage of drone-eligible locations
V_d	2	Drone speedup factor
$it_{restart}$	100.000	Amount of iterations of a random restart
it_{min}	100.000	Minimum iterations in a level of the hierarchy
it_{max}	1.000.000	Maximum iterations in a level of the hierarchy
it_{imp}	30.000	Maximum iterations without improvement in a level of the hierarchy
dp	10%	Percentage diversity contribution in biased fitness
CO	10%	Percentage of iterations that perform cross-over

In tabel 7.1 wordt een overzicht gegeven van alle variabelen die nodig zijn voor het uitvoeren van de tests, we hebben deze zoveel mogelijk proberen reduceren en onze default waarden gegeven die we gebruiken in de tests. Wanneer we vergelijken met waarden uit andere papers hebben we telkens wel met de doelfuncties van de referentie gewerkt zodat we rechtstreeks kunnen vergelijken. Dit zijn de variabelen van onze eigen testen, hierbij zullen we ook telkens de resultaten met de methodiek van Salue et al [46] erbij zetten zodat onderzoek die met zijn oplossingsmethodiek verder werkt ook vergeleken kan worden.

7.4 Performance van HGSD en drones in cVRP

7.4.1 Genetisch algoritme performance op cVRP

Een maatstaaf voor het bekijken van de kwaliteit van ons algoritme is het testen op cVRP. We kunnen namelijk in ons algoritme het aantal drones gelijkstellen aan 0 waardoor het algoritme standaard VRP oplossingen zal genereren. Deze kunnen we vergelijken met de cVRP oplossingen. Dit is slechts een benadering aangezien in ons algoritme geen rekening gehouden wordt met de capaciteit per wagen en wij dus mogelijks over capaciteit kunnen gaan. Ook kan ons algoritme optimaliseren naar makespan of leveringen/uur, maar minimaliseren we bij cVRP de afstand. Goede routes zullen op verschillende metrieken nog steeds degelijke resultaten halen. Hierdoor lijkt deze test nog steeds relevant om te doen.
Wanneer we echter een fitness functie gebruiken die ook de afstand minimaliseert in ons algoritme merken we dat het ontbreken van de capaciteit een grote rol speelt. Zo halen we bij het minimaliseren van de afstand een betere score dan de beste gevonden oplossing, maar doen dit door serieus over capaciteit te gaan. Hierdoor geeft dit een vertekend beeld terwijl bij de metrieken die wij gebruikt hebben, makespan en leveringen/uur, we in de meeste gevallen de capaciteit niet overschrijden.
We kijken bij onze gevonden oplossingen naar de som van de totale afstanden van alle routes. Hiernaast kijken we ook naar de leveringen/uur metriek wat het aantal pakjes is wat geleverd wordt per uur. En uiteindelijk kijken we ook naar de makespan. Bij onze oplossing is voor het optimaliseren van van alle metrieken telkens dezelfde fitness functie gebruikt die de makespan minimaliseert zoals besproken in sectie 4.5. De gebruikte parameters voor het berekenen van de leveringen/uur en makepan zijn te vinden in tabel 7.1 en de leveringen/uur is berekend zoals beschreven in sectie 2.5.4 en van de makespan volgens algoritme 16.

In tabel 7.2 zien we onze resultaten van HGSD ten opzichte van de best known solution van het cVRP algoritme. We hebben de HGSD algoritmes 1u laten draaien op de HPC met 16 cores en 2 Gb memory en de parameters uit tabel 7.1. De best known solution van het cVRP probleem minimaliseert echter wel de totale afstand zonder de capacity contraint van wagens te overschrijden. In onze oplossing

Tabel 7.2: HGSD met geen drones, performance tegenover de best known solution van cVRPD

Instance	BKS distance	HGSD distance	BKS pak/uur	HGSD pak/uur	BKS makespan	HGSD makespan
X-n139-k10	13596	14149	243,244	281,16	1701,99	1472,49
X-n219-k73	117601	116828	234,77	240,142	2785,72	2723,39
X-n275-k28	21250	23171	600,10	755,79	1369,77	1087,6
X-n439-k37	36410,3	39547	849,93	1001,14	1546	1312,5
X-n513-k21	24224	27341	727,07	1049,99	2112,59	1462,87
X-n627-k43	62199	73997	658,54	669,97	2851,75	2803,12
X-n716-k35	43420	52851	974,31	1085,36	2201,56	1976,3
X-n837-k142	193785	203480	1038,75	1072,84	2414,45	2337,73
X-n957-k87	855117	846542	1574,48	1656,45	1821,56	1731,42
X-n1001-k43	72404	83692,9	1050,2	1182,19	2856,6	2537,65

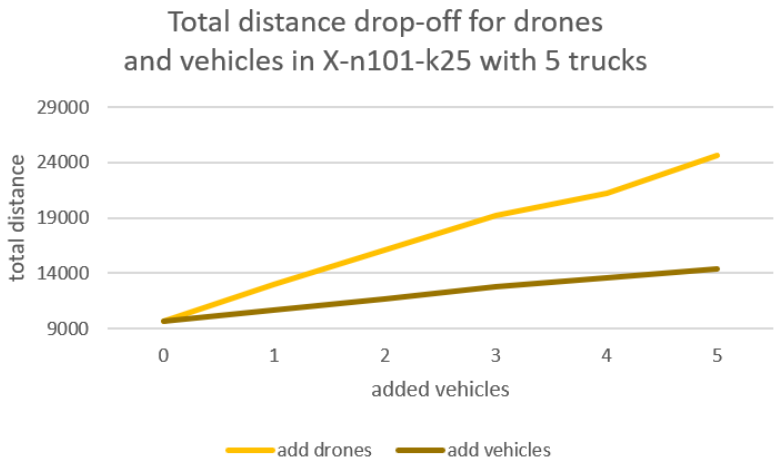
optimaliseren we naar de makespan van het probleem. We zien wel duidelijk een verschil hierin, zo zien we dat bij de totale afstand (distance), de BKS bijna altijd beter zal zijn. We zien hierbij wel veel verschil tussen de verschillende probleem instanties, wanneer er bijvoorbeeld veel restrictie is door capaciteiten dan doet HGSD het soms bijna even goed als de BKS of soms zelfs beter zoals in X-n219-k73. In problemen waar capaciteit een minder grote restrictie is zien we echter dat de verschillen met afstand soms kunnen uitlopen tot 15-20%, de leveringen/uur en makespan waarnaar geoptimaliseerd is bij HGSD is wel voor alle instanties beter dan de BKS van totale afstand, maar deze was hier natuurlijk niet naar geoptimaliseerd, maar dit toont wel dat een zeer goede oplossing, zoals de BKS of een oplossing die manueel gemaakt zou worden nog steeds slechter presteert dan onze oplossing. Bij makespan waar we uiteindelijk naar maximaliseren zien we verschillen van rond de 2-25% met zelfs uitzonderingen van 44%. Bij de leveringen/uur zijn de verschillen wat kleiner tussen 2-17% gemiddeld.

7.4.2 Impact van drones op het VRP

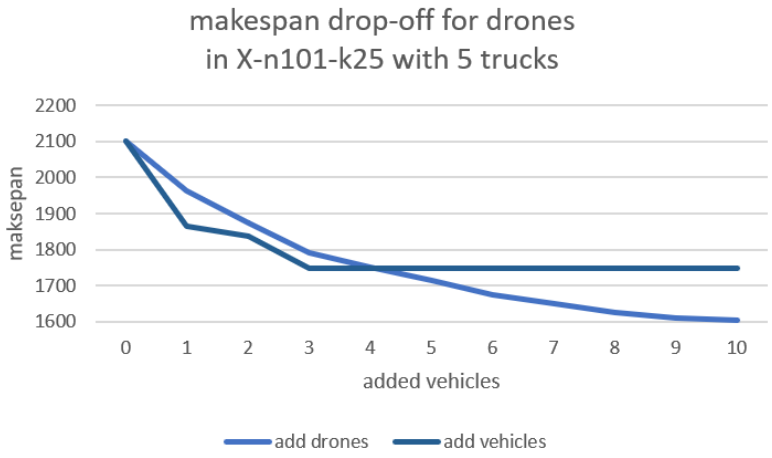
In deze sectie vergelijken we wat de impact is van het toevoegen van drones aan het VRP. We proberen dit op een objectieve manier te doen, die zo nauw mogelijk aansluit bij de realiteit. Hiervoor is het belangrijk dat we ook de variabelen en penalty's in acht nemen die verschillen bij de beide soorten voertuigen. De exacte constanten die we gebruikt hebben staan in tabel 7.1. We gebruiken hierbij ons standaard HGSD algoritme en kijken hoe drones presteren tegenover wagens en kijken welke impact drones hebben op het VRP en hoe dit schaalt en hoe we drones het beste kunnen benutten. We hebben dit getest op een vrij kleine instantie met een centrale depot, namelijk X-n101-k25, en hier het aantal wagens en drones gevarieerd en de verschillen in verschillende metrieken bekeken.

Voor de afstand hebben we eerder al besproken dat het toevoegen van drones enkel nadelig zal zijn voor de totale afstand en we hierdoor dus geen verbeteringen zullen kunnen doen aan de totale afstand. In figuur 7.3 zien we dat door het toevoegen van drones de totale afstand ongeveer lineair zal stijgen. Als we dit vergelijken met toegevoegde wagens zien we dat drones toevoegen een pak meer zal stijgen dan wagens.

Wanneer we de makespan bekijken op figuur 7.4 zien we iets interessant, initieel zal het beter zijn wagens toe te voegen, maar na verloop van tijd komen we aan de best mogelijke makespan voor wagens (heen en terug van verste locatie met wagen) en zal extra wagens toevoegen de makespan niet verlagen. Terwijl drones door hun hogere snelheid wel nog verder kunnen verlagen. We zien zowel bij drones als wagens dat de winst die we halen uit het toevoegen afneemt naar gelang we meer toevoegen. Zo zien we initieel dat één wagen toevoegen ongeveer evenveel effect heeft als het toevoegen van 2 drones, maar bij 4 wagens toevoegen is dit al bijna even efficiënt als 4 drones toevoegen, maar dit is uiteraard zeer afhankelijk van de situatie. Over het algemeen als routes nog veel locaties/route hebben zal een



Figuur 7.3: Toename in totale afstand wanneer we drones en wagens in het probleem toevoegen



Figuur 7.4: Daling in makespan wanneer we drones en wagens toevoegen aan het probleem

wagen toevoegen voor een grotere verbetering zorgen, maar de drop-off bij wagens is veel sneller dan bij drones. Hierdoor zal op termijn drones toevoegen beter worden, maar dit is zeer afhankelijk van de instantie. Er valt dus geen algemene ratio te geven op hoeveel keer wagens beter zijn dan drones, dat hangt ook van allerlei andere factoren af zoals de hoeveelheid wagens en drones die al aanwezig zijn, hoe dicht locaties bij elkaar liggen, het voordoen van clustering of aparte locaties, ...

Als we de kost proberen minimaliseren in ons voorbeeld door aan drones en wagens een realistische reiskost te hangen dan is dit verschil zo groot dat er bijna uitsluitend voor drones gekozen zal worden wanneer geen extra constraints genomen worden. Uit verschillende studies blijkt namelijk dat drones een reiskost hebben van 0,03$/km en wagens een kost van 1,25$/km we gebruiken hierbij de waarden van Nguyen et al. [50]. Dit is een factor 41,67 groter, dus ook al is onze totale afstand een pak groter, zoals te zien in tabel 7.3, is dit nog lang geen factor 41 groter waardoor we zonder constraints altijd alle locaties door drones zullen laten doen die elegible zijn.

Tabel 7.3: Effect van populatiegrootte op makespan en uitvoeringstijd op X-n139-k10 met 5 trucks en 5 drones

population size	makespan 100K	time 100k (s)	makespan 5 min	time 5 min (s)
1	2176,85	4,30	2045,03	300
2	2237,71	4,53	2016,55	300
3	2234,25	5,20	1999,46	300
5	2253,37	5,23	1970,89	300
10	2260,85	5,12	2017,86	300
15	2271,62	5,94	2011,4	300
20	2288,87	6,15	2018,17	300
50	2282,1	9,23	2038,38	300
100	2282,88	15,81	2053,56	300

7.5 Evaluatie HGSD op PDSVRP

Onze testmethode is hoofdzakelijk gemaakt voor PDSVRP waarbij naar de makespan geoptimaliseerd wordt, maar aangezien HGSD puur een slimme manier is om snel en efficiënt oplossingen van PDSVRP te manipuleren kunnen we met een andere fitness functie ook naar andere metrieken optimaliseren. We hebben de fitnes functie, evaluatie en contraints van enkele papers geïmplementeerd en op deze manier rechtstreeks met deze papers vergeleken. Daarnaast hebben we ook voor de oude en nieuwe oplossingen ook met onze eigen test/evaluatie methode geëvalueerd voor mocht iemand later willen vergelijken met de paper dat dit ook makkelijk kan gebeuren.

7.5.1 Analyze van HGSD parameters

HGSD en PDSVRP zijn afhankelijk van een aantal parameters, een groot deel van deze parameters werden reeds toegelicht in tabel 7.1. Een groot deel van deze parameters werden gekozen op basis van parameters in de literatuur en intuïtie, maar deze kunnen een grote invloed hebben op het algoritme. Daarom hebben we hier enkele experimenten opgesteld die aantonen welke verschillen er zijn bij het aanpassen van variabelen op de uitkomst. Op deze manier kunnen we op een empirische wijze bekijken welke parameters het beste zijn voor welke problemen en wat de relatie hiertussen is.

Population size

Een belangrijke parameter in elk genetisch algoritme is de population size en survival rate, dit is het aantal oplossingen die we bijhouden in onze populatie en hoeveel oplossingen elke generatie over gaan naar de volgende generatie. Zoals besproken in sectie 6.2 gebruiken wij geen survival-rate maar wel een populatie vol survivors waarbij we telkens de slechtste in-place muteren aangezien dit voor ons een stuk efficiënter is. We hoeven dus geen rekening te houden met de survival-rate, maar de grote van onze populatie blijft dus wel nog een belangrijke parameter. Deze waarde geeft een trade-off tussen exploratie en efficiëntie, wanneer de populatie groter is zal deze namelijk minder bezig zijn met de goede oplossingen maar wel aan meer verschillende oplossingen werken. We hebben dan ook een test opgesteld die dit aantoont, eerste hebben we de algoritmes 100k iteraties laten lopen met verschillende populatie grote, en hierna hebben we de populaties 5 min lang laten lopen met verschillende populatie grote. We bekijken dan het verschil in executie tijd en het verschil in makespan. De resultaten hiervan staan in tabel 7.3. We hebben de HGSD algoritmes laten draaien op de HPC met 16 cores en 2 Gb memory en de overige parameters uit tabel 7.1 en vervolgens de mediaan genomen van 10 runs.

Als we kijken naar de resultaten in tabel 7.3 dan zien we dat onze intuïtie klopt, na de eerste 100k stappen scoren kleinere populaties beter. De spread die we kregen bij de oplossingen van de verschillende runs hadden ook een grote spreiding, wat bij de 5 min minder het geval was. In de eerste

Tabel 7.4: Effect van cross-over percentages op de makespan van X-n139-k10 met 5 wagens en 5 drones

cross-over (%)	makespan	time (s)
0	2004.06	300
2	2010.63	300
5	2005.11	300
10	1996.93	300
15	2012.86	300
20	2030.78	300

100k iteraties draait het dus nog voor een groot deel om geluk om goeie random mutaties te doen en liefst op de beste locatie. Daarom presteren kleine populaties hier beter omdat de kans dat we aan de beste oplossingen verder werken daar een pak groter is.
Als we dan kijken naar 5 min dan zien we dat grotere populaties beter beginnen scoren. In dit geval zien we een daling richting een populatie 5 en hierna opnieuw een stijging. Dit valt te verklaren doordat populatie 5 hier de beste afweging tussen efficiëntie en exploratie heeft voor 5 min. Maar stel dat we langer lieten zoeken is de kans groot dat grote populatie gelijkaardige of zelfs betere oplossingen zullen vinden. Dit hangt ook veel af van de grote van het probleem, als we een groter probleem hadden genomen was na 5 minuten hoogst waarschijnlijk de laagste populatie nog steeds de beste. Daarom heeft de probleem grote en mogelijke rekentijd dus ook een grote invloed op wat de ideale populatie grote is. Over het algemeen zien we dat 5 tot 15 de beste resultaten geeft, daarom onze keuze voor 10. Door de spreiding van de resultaten zien we wanneer we geluk hebben en goeie mutaties doen, we hier goede oplossingen krijgen, en andere keren wanneer we geen geluk hebben halen we slechte resultaten, daarom nemen we altijd de mediaan tussen 10 runs.

Cross-over percentage

Een andere belangrijke variabele voor het genetische algoritme is het percentage van de iteraties dat we een cross-over doen. We hebben dan ook verschillende percentages cross-over getest en vergeleken. De resultaten staan in tabel 7.4. We hebben de HGSD algoritmes laten draaien voor 5 minuten op de HPC met 16 cores en 2 Gb memory en de overige parameters uit tabel 7.1 en vervolgens de mediaan genomen van 10 runs.

In tabel 7.4 zien we dat bij lage percentages van cross-over we initieel vrij goeie oplossingen krijgen. Deze spread die we zien op de oplossingen is echter wel nog vrij groot, daarom zien we wat inconsistenties in de tabel, we hebben echter de mediaan van 10 runs van 5 minuten gebruikt om dit te proberen vermijden. Tot en met 10% cross-over blijven onze oplossingen vrij goed, maar vanaf we hoger gaan dan 10% zien we duidelijk een toename in de makespan. Daarom hebben we 10% genomen als optimale parameter. Als we kijken hebben we met 7 mutaties en 1 cross-over een gelijke verdeling met $100\%/8 = 12,5\%$ dus met cross-over op 10% zullen alle mutaties $90\%/7 = 12,86\%$ van de iteraties gekozen worden. Ook dit is opnieuw een afweging, als we onze cross-over laag houden zullen andere mutaties meer tijd krijgen. Momenteel doen we na een cross-over ook nog geen reparatie stap achteraf wat in veel algoritmes zoals Vidal et al. [66] wel gedaan wordt. Mogelijks kunnen we met een reparatie stap achteraf de effectiviteit van onze cross-over beter maken.

Drone speed

Een belangrijke factor in VRP met drones is hoe we het verschil tussen drones en vehicles zullen weergeven. We hebben er in deze scriptie voor gekozen om voor de afstand voor beide voertuigen de Euclidische afstand te nemen, om het verschil in snelheid aan te geven gebruiken we een drone speed factor. Deze geeft aan hoeveel sneller drones zijn dan wagens. In de literatuur wordt deze factor vaak op 2 gezet in combinatie met Manhattan distance [46] [55]. We hebben echter besproken dat

Tabel 7.5: Invloed van drone-snelheidsfactor op makespan en drone-percentage van X-n139-k10 met 5 wagens en 5 drones

Drone speed	makespan	drone %
1	1996.93	13.14
2	1856.81	21.17
3	1718.42	27.01
4	1655.80	35.04
5	1563.16	40.88

Tabel 7.6: Effect van diversity contribution op makespan van X-n139-k10 met 5 wagens en 5 drones

diversity percentage (%)	makespan
0	1847.48
1	1895.31
2	1911.64
5	1870.67
10	1856.41
20	1887.22
30	1824.77

Euclidische afstand gebruiken voor beide en een drone speed van 3 een meer realistische waarde is. Maar omdat in de literatuur het meeste 2 gebruikt wordt hebben we dit ook gedaan voor onze tests. Hier bekijken we de impact op de drone speed op de makespan en het percentage van de locaties die door drones bezocht wordt.

De resultaten van de experimenten die we hebben gedaan staan in tabel 7.5. We hebben de HGSD algoritmes laten draaien op de HPC gedurende 5 minuten met 16 cores en 2 Gb memory en de overige parameters uit tabel 7.1 en vervolgens de mediaan genomen van 10 runs. We zien hier duidelijk dat de drone speed een grote invloed heeft op zowel de makespan als het percentage locaties die door drones behandelt wordt. Dit is bijna een lineaire relatie tussen de drones speed en de makespan, hierdoor is het sneller maken van drones dus zeker een goeie inspanning om PDSVRP efficiënter te maken. Dit is hoogst waarschijnlijk waarom prototype drones van Amazon [3] en Wing [70] hoge snelheden van 80-100 km/h halen. hetzelfde zien we bij het percentage van de locaties behandelt door drones, ook hier is een serieuze stijging die bijna lineair is.

Diversity contribution percentage

Bij het berekenen van de biased fitness hebben we een constant percentage waarmee we bepalen hoeveel de diversiteits contributie meetelt in de berekening van de biased fitness functie. We geven nogmaals de formule van de biased fitness functie:

$$BF(P) = (1 - dp) \cdot fitness(P) + dp \cdot fitness(P) \cdot \Delta P$$

Hier stelt dp het diversiteit percentage voor ofwel het percentage van de biased fitness functie dat de fitness meetelt tegenover de diversiteit contributie. Wanneer deze laag is dan kijken we vooral naar de beste oplossingen en werken we hierop verder, maar als deze hoog is dan kijken we meer naar de diversiteit en geven we andere slechtere oplossingen ook meer de kans om te trainen.

We hebben in de experimenten de HGSD algoritmes laten draaien voor 5 minuten op de HPC met 16 cores en 2 Gb memory en de overige parameters uit tabel 7.1 en vervolgens de mediaan genomen van 10 runs. In tabel 7.6 zien we de impact die het biased percentage heeft op de uiteindelijke oplossing.

Tabel 7.7: Aantal leveringen door drones vs wagens in X-n139-k10 met 5 wagens en 5 drones

Instance	#D	#V	#drone delivered	#wagen delivered	drone %
X-n139-k10	5	5	19	119	13.77%
X-n439-k37	19	18	42	396	9.58 %
X-n513-k21	11	10	61	451	11.91 %
X-n627-k43	22	21	57	569	9.10 %
X-n716-k35	18	17	85	630	11.89 %
X-n837-k142	71	71	192	644	22.97 %
X-n957-k87	44	43	147	809	15.38 %
X-n1001-k43	22	21	101	899	10.10 %

We zien eigenaardig genoeg geen grote verschillen tussen de oplossingen en de meeste verschillen die we zien lijken vooral ruis te zijn. Het diversity percentage lijkt dus op het eerste zicht geen grote invloed te hebben na een groot aantal iteraties. We zouden dit echter wel verwachten aangezien dit normaal onze diversiteit in de populatie zou moeten versterken. We zien echter dat na veel iteraties zelfs met een hoge diversity contributie de diversiteit snel uit onze populatie is. Verder onderzoek zou kunnen uitwijzen hoe dit komt en of dit misschien een fout kan zijn in onze implementatie.

Aantal leveringen door drones vs wagens

Het is ook een interessant te bekijken hoe de verdeling van het werk nu zit tussen drones en wagens. We zien wanneer we evenveel drones en wagens gebruiken en als verschil tussen drones en wagens nemen dat drones dubbel zo snel zijn waar wel slechts 1 pakket kunnen leveren per keer, en wagens meerdere pakketten per keer. De resultaten van onze experimenten staan in tabel 7.7. We hebben de HGSD algoritmes laten draaien op de HPC gedurende 5 minuten met 16 cores en 2 Gb memory en de overige parameters uit tabel 7.1 en vervolgens de mediaan genomen van 10 runs. We zien dat gemiddeld 9-15% van de locaties door drones gedaan zal worden. Dit hangt echter veel af van de probleem instantie. Wanneer we bijvoorbeeld veel eenzame locaties hebben zonder dichte buren kan dit percentage verhogen. Ook zien we dat een groot verschil in afstand dit percentage kan beïnvloeden. Zo zien we bijvoorbeeld bij instantie X-n837-k142 een outlier van 22.97% dit komt omdat de map hier een vierkant is met uniform verdeelde punten waar de depot hier bijna in een hoek aan een ver uiteinde zat, en sommige locaties dus zeer ver lagen. Deze verre locaties in de tegenovergestelde hoek werden dan vaak gedaan door de drones om zo de makespan te drukken die hier door het grote aantal wagens en de verre locaties zwaar afhing van de verste locatie gedaan door een wagen.

Amount of iterations

Andere parameters in tabel 7.1 die we onderzocht hebben zijn de iteraties. Tijdens het onderzoek zijn de huidige waarden empirisch naar boven gekomen als de beste voor de meeste instanties die we gebruiken. De it_{min}, it_{max} en it_{imp} parameters, zijn in principe op termijn niet echt van zeer groot belang, deze bepalen hoe snel we overgaan naar het volgende level in onze hiërarchie, maar in onze parallelle implementatie doet het er vooral toe dat een thread altijd kan blijven verder werken aan de beste populatie terwijl andere threads nieuwe oplossingen zoeken voor het volgende level te vullen. Maar eenmaal onze head thread altijd kan blijven doorzoeken aan de beste populatie zonder te moeten wachten op nieuwe oplossingen kunnen we hier in principe niet veel beter doen. Een parameter die echter wel belangrijk is en in ons model misschien te simpel voorgesteld wordt is $it_{restart}$. Deze bepaalt namelijk hoe goed onze oplossingen zullen zijn die we invoegen elk level van de hiërarchie. momenteel staat deze op 100k omdat dit een goede hoeveelheid is voor de meeste kleine problemen en meestal vrij snel verloopt waardoor we nooit in problemen komen met het aantal oplossingen. Wanneer we echter grote problemen hebben zijn deze random restarts vaak nog van slechte kwaliteit na 100k iteraties en

na het lang lopen van ons algoritme vaak niet meer vergelijkbaar van kwaliteit met onze huidige beste. Daarom zal het toevoegen van deze slechte oplossingen dan bijna nooit een effect geven. We zouden dan in principe beter onze random restarts langer laten trainen zodat het toevoegen ervan meer nut heeft. We hebben hier echter niet veel tijd meer gevonden om dit te onderzoeken.
Een mogelijke oplossing zou kunnen zijn om random restarts te trainen tot ze aan een bepaalde kwaliteit komen in vergelijking met de huidige beste oplossing. We kunnen bijvoorbeeld blijven trainen zolang we niet op 10% van de beste oplossing liggen en maximaal it_{imp} iteraties geen improvement gemaakt hebben. Op deze manier weten we dat de oplossingen die we toevoegen altijd van degelijke kwaliteit zullen zijn en nuttig zijn om toe te voegen aan de beste populatie. In future work kan dit zeker onderzocht worden hoe we dit het beste aanpakken.

7.5.2 HGSD PDSVRP benchmarks

Volgens de testmethode hierboven beschreven hebben we experimenten gedaan op enkele test instanties van Uchoa et al. [61] uit CVRPLIB [71]. Dit werd gedaan met Euclidische afstand voor zowel drones als wagens. We hebben tests gedaan met zoowel 80% als 100% drone toegankelijke locaties. Deze werden gekozen aan de hand van het de capaciteit, waarbij de locatiets met 20% hoogste capaciteit drone ontoegankelijk gezet werden.

Tijdens het testen werd echter wel duidelijk dat sommige instanties niet zeer geschikt waren voor PDSVRP, zo zagen we dat instanties met een hoog aantal koeriers en een verre (niet centrale) depot vaak als beste makespan een route hadden van een wagen die naar de verste ontoegankelijk locatie en terug ging. Dit is echter wel een bewezen beste oplossing van het probleem en hebben we ook aangeduid met een asterix(*) in de oplossingen, maar dit is natuurlijk geen mooie oplossing van het probleem. De mooiste oplossing komen van oplossingen met weinig koeriers of een centrale depot, we hebben dan ook vooral tegen dit soort instanties getest.
De oplossingen worden gegeven in tabel 7.8. We hebben de HGSD algoritmes laten draaien op de HPC gedurende 1 uur met 16 cores en 2 Gb memory en de parameters uit tabel 7.1. Een voorbeeld van een grote oplossing wordt gegeven in figuur **??**. Op deze figuur zien we hoe bij instanties die geen centrale depot hebben de routes veel kunnen verschillen van aantal locaties. We zien bijvoorbeeld sommige routes die bijna in een rechte lijn naar de verste locaties gaan daar enkele locaties doet en terug keert. Terwijl andere een pak meer locaties doen. Over het algemeen zien we wel efficiënte routes terugkomen naar de uiteinden van de kaart om daar terug te keren. We zien ook dat er op sommige plaatsen nog ruimte voor verbetering mogelijk is, maar dit is dan ook een van de grootste instanties waarop we testen. Bij de drones zien we duidelijk dat de uiterste locaties verkozen worden om de routes niet helemaal naar daar te laten gaan, maar ook de dichtste omdat drones deze efficiënt kunnen doen.

Bij het kiezen van de test instanties hebben we enkele willekeurig genomen die duidelijk de problemen aangeven met sommige VRP instanties, zoals X-n837-k142 en X-n916-k207, maar andere hebben we proberen nemen met een centraal depot en niet te veel koeriers. Op deze manier hebben we het meest realistische scenario waarin drone delivery het beste gebruikt kan worden en deze geven de mooiste oplossingen tegenover sommige andere cVRP instanties. Dit zijn echter de nadelen die we hebben door het hergebruiken van test instanties.

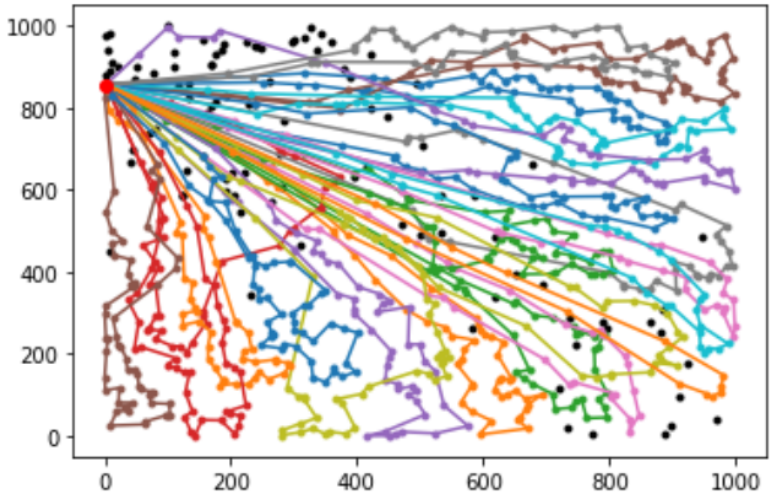
Experimenten op Raj et al. [55]

We hebben ook vergeleken met de PDSVRP instanties van [55]. Hier werden TSPLIB instanties gebruikt als benchmarks en was het aantal wagens beperkt tot 3 en het aantal drones tot 6. Deze gebruikte de manier van evalueren zoals Salue et al. [46], deze staan reeds beschreven in sectie 7.2.

Onze resultaten hiervan staan in tabel 7.9. We hebben de HGSD algoritmes laten draaien op de HPC gedurende 1 uur met 16 cores en 2 Gb memory en de parameters uit tabel 7.1. We hebben hierbij exact dezelfde constraints parameters en fitness functie gebruikt als in de paper van Raj et al.

Tabel 7.8: Resultaten PDSVRP

Instance	#D	#V	HGSD 80%	HGSD 100%	gap %	HGSD Manh
X-n139-k10	5	5	1844.74	1795.57	-2.66 %	2164
X-n275-k28	14	14	1050.08	1040.16	-0.94 %	1528
X-n439-k37	19	18	1309.90	1290.09	-1.53 %	1856
X-n513-k21	11	10	1757.50	1710.78	-2.73 %	2208
X-n627-k43	22	21	2801.00	2563.95	-9.24 %	3960
X-n716-k35	18	17	1950.77	1786.24	-9.21 %	2718
X-n783-k48	24	24	2487.93	2274.74	-9.37 %	3592
X-n837-k142	71	71	2287.9*	1954.04	-17.08 %	3282
X-n916-k207	104	103	2680.47*	2026.65	-32.26 %	3920
X-n957-k87	44	43	1640.85	1569.79	-4.53 %	2436
X-n979-k58	29	29	2697.49	2600.55	-4.04%	3992
X-n1001-k43	22	21	2542.55	2405.77	-5.36 %	3574



Figuur 7.5: Oplossing voor X-n1001-k43 met 80% drone toegankelijke locaties, met zwarte locaties behandeld door drones

Tabel 7.9: PDSTSP & PDSVRP resultaten voor TSPLIB-instanties van Salue et al. [46]

		1			2			3		
Instance	#D	BKS	HGSD	gap %	BKS	HGSD	gap %	BKS	HGSD	gap %
att48	2	28686	**25695.5**	-10.42	17032.0	17196	0.96	14062	**13340**	-5.13
	4	28610	**20276**	-29.12	16500.0	**13811.6**	-16.29	13394	**12990**	-3.02
	6	28610	**20182**	-29.45	16500.0	**13246**	-19.72	13394	**12624**	-5.75
berlin52	2	5290.65	5330	0.74	3328.2	3348.78	0.62	2995.0	**2945**	-1.67
	4	5190.00	5190	0.0	2995.0	**2945**	-1.67	2625.0	2685	2.29
	6	5190.00	**5180**	-0.19	2995.0	**2945**	-1.67	2625.0	2675	1.90
eil101	2	456.00	473	3.73	305.4	**302**	-1.11	235.0	**229**	-2.55
	4	346.68	395	13.94	253.7	**252**	-0.67	200.1	204	1.95
	6	314.0	395	25.79	215.3	224	4.04	181.0	186	2.76
gr120	2	1188.51	1212	1.97	764.0	**754**	-1.31	597.0	**584**	-2.18
	4	946.04	1000	5.70	646.0	**644**	-0.31	528.0	**522**	-1.14
	6	851.4	927	8.88	581.2	**580**	-0.21	527.2	**476**	-9.71
pr152	2	70244	73630	4.82	48967.0	**41566.6**	-15.11	48135	**33590.1**	-30.22
	4	59772	64773.1	8.37	52353.0	**38672.6**	-26.13	43024	**32507.9**	-24.44
	6	56262	62202	10.56	50854.4	**35509**	-30.18	41324	**30807**	-25.45
gr229	2	1673.72	1742.09	4.08	1403.7	**1008.54**	-28.15	1145.4	**698.94**	-38.98
	4	1518.62	**1484.02**	-2.28	1346.2	**916.94**	-31.89	1132.3	**647.9**	-42.78
	6	1467.76	**1362.46**	-7.17	1327.6	**867.48**	-34.66	1130.0	**631.54**	-44.11

[55]. We zien dat we op de grote instanties een enorme verbetering kunnen doen op de huidige beste oplossingen, maar zeker bij de instanties met 1 drone wat dus eigenlijk een PDSTSP is hebben we soms slechtere resultaten. Bij de instanties met meerdere drones halen we wel consistent betere resultaten.

7.5.3 Vergelijking met PDSmTSP

PDSmTSP werd voorgesteld in Salue et al. [58] en is een probleem dat nauw aansluit bij het probleem van PDSVRP, het enige grote verschil dat we hebben is dat bij PDSmTSP we geen limiet zetten op de capaciteit van de wagens. Hierdoor kunnen deze dus zeer grote nummer aannemen. Maar aangezien we de makespan zo klein mogelijk houden heeft dit als gevolg dat we dus ook het werk zo goed mogelijk verdelen en dus zelden in problemen komen met capaciteit. Oplossingen voor PDSVRP zijn dan ook altijd correcte oplossingen voor PDSmTSP, hierdoor kunnen we dus onze oplossingen vergelijken met elkaar. Er wordt echter wel een andere manier van het berekenen van de makespan gebruikt. Zo zal in Salue et al. [58] de Manhattan distance gebruikt worden voor de afstand van een wagen, maar is voor de rest de reis kost van drones en wagens gelijk. Maar hier werden ook cVRP benchmarks gebruikt met aanpassingen aan de grootte van de vloot, deze werd aangepast door drones te introduceren, dit werd gedaan met volgende formule $Trucks = \lceil \frac{fleetSize}{2} \rceil$ en $Drones = \lfloor \frac{fleetSize}{2} \rfloor$. Alle locaties werden toegankelijk voor drones beschouwd. We hebben voor het PDSmTSP probleem enkel de resultaten van Salue et al [58] gevonden en zien deze dus als de huidige best known solutions BKS.

De resultaten zien we in tabel 7.10. We hebben de HGSD algoritmes laten draaien op de HPC gedurende 1 uur met 16 cores en 2 Gb memory en de parameters uit tabel 7.1. We zien dat we op bijna alle instanties een verbetering hebben weten te halen met ons algoritme, zeker wanneer de test instanties groot worden zien we dat onze oplossing hier een stuk beter kan worden en de gap ook vergroot.

7.5.4 HGSD op realistische large-scale VRP instanties.

We hebben onze genetische methode zo ontwikkeld dat deze ook voor large-scale VRP instanties gebruikt kan worden. De enige aanpassing die we hiervoor moeten doen is mutaties met een slechte

Tabel 7.10: Resultaten PDSmTSP

Instance	n	#D	#V	BKS	HGSD	gap
CMT1	50	3	2	166	166	-0.0 %
CMT2	75	5	5	130.23	128.774	-0.95 %
CMT3	100	4	4	184	184	-0.0 %
CMT4	150	6	6	160.38	154	-4.14 %
CMT5	199	9	8	138	134	-2.99 %
E-n51-k5	50	3	2	168	166	-1.2 %
E-n76-k8	75	4	4	154	152	-1.32 %
E-n101-k8	100	4	4	184	184	-0.0 %
M-n151-k12	150	6	6	154	152	-1.32 %
M-n200-k16	199	8	8	144	138	-4.35 %
P-n51-k10	50	5	5	111.07	110	-0.97 %
P-n55-k7	54	4	3	126	126	-0.0 %
P-n60-k10	59	5	5	114	114	-0.0 %
P-n65-k10	64	5	5	126	124	-1.61 %
P-n70-k10	69	5	5	128	128.253	+0.2 %
P-n76-k5	75	3	2	200	200	-0.0 %
P-n101-k4	100	2	2	342	342	-0.0 %
X-n110-k13	109	7	6	1864	1812	-2.87 %
X-n115-k10	114	5	5	2258	2150	-5.02 %
X-n139-k10	138	5	5	2492	2376	-4.88 %

Tabel 7.11: large-scale PDSVRPD resultaten

Instance	n	#D	#V	makespan
Leuven2	4000	35	35	3319.08
Antwerp2	7000	60	60	4466.41
Ghent2	11000	95	95	3386.71

tijdscomplexiteit zoals de cross mutatie laten vallen, of een efficiëntere heuristiek toekennen. Verder kunnen we zien uit figuur 4.15 dat er geen andere mutaties zijn die eruit springen op vlak van slechte performance, daarom hebben we enkel de cross heuristiek aangepast. We zien echter wel dat deze grotere instanties niet kunnen draaien op enkele minuten tijd en hebben daarom de looptijd ervan verhoogd naar 8 uur. Wat eigenlijk een nacht voorstelt waarin het algoritme zou kunnen draaien. Voor de rest hebben we wel gezien dat sommige van onze heuristieken niet echt aangepast zijn op grote instanties en vooral goed op kleine tot middelgrote instanties werken. Onze tests zijn gedaan op de large-scale instanties van de set van realistische VRPs van Vlaamse steden uit Arnold et al. [4] met 80% drone toegankelijke klanten, Euclidische afstand voor zowel drones als wagens en een drone speed van 2.

De resultaten zijn te zien in tabel 7.11. We hebben de HGSD algoritmes laten draaien op de HPC gedurende 8 uur met 32 cores en 4 Gb memory en de parameters uit tabel 7.1, met de drone toegankelijke locaties op 100%. We zijn dat we reeds zeer degelijke resultaten kunnen bekomen voor de omvang van deze problemen. Als we deze oplossingen bekijken zien dit er ook vrij logische oplossingen uit. We zien ook na 8 uur dat er nog steeds elk level van de hiërarchie een verbetering komt. Als we het algoritme dus langer laten lopen kunnen we nog betere oplossingen behalen.

7.6 HGSD op PDSTSP

Aangezien dat PDSVRP een generalisatie is van PDSTSP kunnen we makkelijk door het aantal wagens op één te zetten ook PDSTSP instanties testen, onze resultaten hiervan staan in de eerste kolom van tabel 7.9. Raj et al heeft een generalisatie gemaakt tegenover de huidige PDSTSP resultaten. Voor deze oplossingen hebben we ook de beste oplossingen genomen tussen de papers van Raj et al. [55], Salue et al. [46] en Dell'Amico et al. [17]. De mindere performance bij sommige instanties met 1 wagen zijn ook makkelijk verklaren omdat veel van onze mutaties tussen verschillende routes zijn en deze dus nutteloos zullen zijn voor PDSTSP en ons algoritme hier niet op aangepast is. Ondanks die zaken weten we wel een verbetering te halen op de grootste instanties. Ook op att48 halen we een grote verbetering, maar dit lijkt ons zeer vreemd aangezien dit zo een kleine instantie is en kan mogelijks een fout zijn. We kwamen hier echter consistent dit resultaat uit waardoor we het laten staan, en ook de evaluatie strategie is meerdere malen gecheckt, we halen ook gelijke ondergrenzen bij andere instanties waardoor de evaluatie wel lijkt te kloppen.

7.7 Benchmarking voor het dynamische VRP

Een grote extra moeilijkheid bij het testen doet zich voor wanneer we het probleem dynamisch maken. Er bestaat namelijk geen standaard benchmark dataset voor het meten van de performantie van het systeem. Er zijn namelijk veel parameters die hier ook een rol spelen. Ten eerste zou de uitvoeringstijd hier een grote rol beginnen spelen, en de volgorde en tijdstippen dat locaties toegevoegd worden. Ook het aantal locaties toegevoegd per tijdseenheid en dergelijke moet in rekening gebracht worden. Daarom is het maken van een dynamische benchmark een zeer lastige taak. De manier waarop dynamische VRP's die online kunnen werken gebenchmarkt worden is meestal offline op een statische manier zoals bij Mavrovouniotis et al. [45]. Hierdoor zien we wel slechts een deel van de picture aangezien bij ons genetisch algoritme bijvoorbeeld normaal een deel van de nodes opgelaten wordt voor een volgende route. Dus als we dit niet meer toestaan zal dit gevolgen hebben voor de performantie. Een andere vorm van benchmark die we kunnen doen is om te kijken hoeveel verlies we hebben aan performantie door het dynamisch uitvoeren van de routes. We kunnen van een standaard benchmark datasets met een deel van de locaties starten en hierna de locaties 1 voor 1 toevoegen gespreid over een bepaalde tijd en ondertussen de koeriers laten rijden. Dan kunnen we zien op het einde van de tijd hoelang we nodig hadden om alle locaties te bezoeken en dat vergelijken met de best known solution wanneer we het statisch oplossen. Een andere manier waarop dynamische VRPs gebenchmark worden is door lateness te introduceren wat een penalty geeft op de uiteindelijke score. In ons geval lijkt een van de beste metrieken het maximaliseren van het aantal locaties bezocht binnen een gegeven tijd zoals bij Ulmer et al. [62] waar het aantal locaties met same-day delivery gemaximaliseerd wordt maar dit komt op hetzelfde neer. Dan dient de benchmark hier wel aan aangepast worden dat we niet altijd alle locaties kunnen bezoeken.

Hoofdstuk 8

Future work

Het onderzoek naar drone delivery en dan specifiek PDSVRP is nog steeds pas kort geleden in gang geschoten. Op het moment van het schrijven van deze paper is de eerste paper over drones in TSP van Murray et al. [48] 7 jaar oud. In deze tijd is er vooral ook aandacht besteed aan het flying sidekick TSP, en staat onderzoek naar PDSTSP nog niet zo ver op bepaalde vlakken. De eerste papers die PDSTSP dan ook gebruikt hebben met meerdere trucks en drones van Ham et al. [28] en Ulmer et al. [62] zijn dan ook nog geen 4 jaar oud en andere papers over dit probleem zitten in journals die nog niet gepubliceerd zijn. Om maar aan te tonen hoe actueel dit onderzoek.

Deze scriptie heeft zich vooral gefocust op PDSVRP, de nieuwe oplossingsmethode hiervoor HGSD. We hebben ook veel ideeën en concepten onderzocht voor een online variant die same-day delivery zou kunnen faciliteren. We hebben deze ideeën echter niet geïmplementeerd en kunnen testen op onze huidige benchmarks. Mocht er een vervolg komen op deze paper zou dit een logische extensie zijn waarbij we de online variant volledig uitwerken en deze implementeren en hier experimenten op uitvoeren. Het zou vooral interessant zijn om een schatting te kunnen maken over hoe groot het verlies aan efficiëntie is wanneer we overgaan naar een same-day delivery setting en te zien hoe onze ideeën presteren.

Een volgende verbetering die nog gemaakt kan worden aan de eindoplossing van HGSD is bij het inplannen van de drones. Hiervoor gebruiken we nu de longest processing time first heuristiek maar dit is slechts een benadering en er bestaan algoritmes die deze inplanning nog efficiënter kunnen doen. Maar deze zijn te intensief om te kunnen gebruiken in een fitness functie van een genetisch algoritme. Maar op onze eindoplossing zouden we wel zo een algoritme kunnen loslaten om nog tijd van de drones te kunnen snoeien. Het vergelijken van onze IPS methode voor drones schedulen en de meer klassieke manier met aparte drone routes die ook gevormd worden met behulp van de local search vergelijken kan ook interessant zijn aangezien in theorie we dan betere oplossingen zouden kunnen bekomen maar wel een pak trager zouden convergeren aangezien we dan tegelijk goeie drone routes en wagen routes moeten construeren.

Een andere verbetering die we nog kunnen doen aan het algoritme is de cross-over, deze is momenteel enkel gedefinieerd over 1 route, maar mochten we dit over meerdere routes kunnen doen en op grotere delen cross-over kunnen definiëren zou dit een grotere impact kunnen hebben.
Sommige heuristieken zijn nu ook generiek en nemen een random grootte van mutatie. We zien echter dat grote mutaties enkel in het begin voor verbetering zorgen maar naar het einde toe zorgen kleine mutaties bijna uitsluitend voor de verbeteringen, daarom zouden we door dit soort zaken te implementeren onze mutaties een pak doeltreffender kunnen maken.

Op vlak van testen en evolueren hebben we in section 7 ook duidelijk kritiek gegeven op de huidige test en evalueer methodes die gebruikt worden in FSTSP en PDSTSP/PDSVRP. We kwamen zelf met

een nieuwe verbeterde aanpak die betere tests geeft en een evaluatie methode die betere resultaten geeft. Er is dus nood aan een review paper die de huidige test/evaluatie methodes vergelijkt en een standaard kiest voor het testen en evalueren van PDSTSP en FSTSP. Want nu zijn er veel verschillende methodes voor problemen die eigenlijk hetzelfde zijn en naar hetzelfde optimaliseren, zeker in PDSTSP, waardoor het lastig is met elkaar te vergelijken. Een dergelijke paper zou ook een benchmark dataset kunnen uitbrengen die specifieke problemen die voor PDSVRP ontworpen zijn kan uitbrengen. Want zelfs met onze methode van het omvormen van CVRPLIB instanties zien we dat veel van deze instanties hier niet op voorzien zijn en dus lelijke oplossingen geven en niet zo nuttige tests blijken te zijn.

Verder zijn er nog heel wat andere local search heuristieken die nog niet uitgeprobeerd zijn op PDSVRP, zeker technieken als destroy/repair methodes en large neighbourhood search zien er veel beloven uit op PDSVRP, maar ook tal van andere methodes kunnen omgevormd worden om ook te werken op PDSVRP. Heel wat methoden en technieken voor TOP en PTP kunnen ook makkelijk gebruikt worden in PDSVRP en dit is voorlopig nog een onuitgeputte bron van verder research die gedaan kan worden.

Verder zijn er nog enkele extensies van het probleem zoals parallel drone scheduling capacitated vehicle routing problem (PDScVRP) waarbij ook elke locatie een demand heeft, PDSVRP met time windows (PDSVRPTW), maar ik denk dat vooral de online variant de interessantste kan zijn aangezien deze enkele interessante eigenschappen heeft zoals besproken in de scriptie.

Hoofdstuk 9

Conclusion

In deze scriptie hebben we onderzoek gedaan naar het het parallel drones scheduling vehicle routing problem, een probleem dat voor het eerst geïntroduceerd werd door Murray et al. [48]. We bespraken het probleem samen met de verschillende varianten die bestaan en de verschillende doelfuncties die ervoor gebruikt worden. Er werd hierna ook een nieuwe methode geïntroduceerd om HET PDSVRP op te lossen. De hybrid genetic search with drones (HGSD) is geïnspireerd op het HGSADC algoritme van Vidal et al. [66] en werd daarvoor grotendeels omgevormd om te werken op PDSVRP. Veel van de belangrijke ideeën en concepten van de paper van Vidal et al. zijn wel gebleven. Ons HGSD algoritme haalt betere resultaten op de PDSVRP benchmarks dan de huidige state-of-the-art oplossing methoden. De focus van de paper ligt voornamelijk op het ontwikkelen van een PDSVRP algoritme dat gebruikt kan worden voor praktische large-scale applicaties van drone delivery. HGSD is dan ook het eerste algoritme dat we gevonden hebben van deze soort dat PDSVRP kan oplossen voor problemen met meer dan 250 locaties in een behoorlijke tijd en kan zelfs large-scale problemen oplossen met 4.000 tot 11.000 locaties. We hebben onze implementatie ook getest tegen de oplossing methodes en benchmarks in de papers van Salue et al. [58], Raj et al. [55] en Dell'Amico et al. [17]. Hierbij hebben we de best known solutions kunnen verbeteren voor de meeste gevallen van PDSmTSP en PDSVRP en ook enkele grote gevallen van PDSTSP.

We hebben onze oplossingsmethode en de implementatie daarvan gedetailleerd beschreven en zelfs een open-source implementatie publiek beschikbaar gesteld voor HGSD op Github.

We hebben hierna ook de mogelijk bekeken om een online algoritme van HGSD te maken dat gebruikt kan worden voor same-day delivery. We bespraken hierbij ook enkele belangrijke struikelblokken voor het oplossen van het dynamische probleem en gaven telkens een oplossing voor de problemen. Er is echter geen implementatie voor dit algoritme voorzien.

We namen de huidige evaluatie en testmethodes van PDSTSP en PDSVRP ook onder de loep en uitten hier ook kritiek op sommige practices en methodes die hier gebruikt worden. We komen dan ook zelf met een nieuwe test- en evaluatiestrategie voor het evalueren van PDSVRP die ons een pak betere probleem instanties opleveren en een objectieve, correcte evaluatie methode van PDSVRP die over het algemeen betere, logische oplossingen zal geven.

Het huidige onderzoek in PDSTSP en PDSVRP is nog steeds slechts enkele jaren oud en staat nog steeds in zijn kinderschoenen. Door het actief onderzoek van grote bedrijven zoals Google, DHL en Amazon zien we dat dit in de academische wereld ook zeer populair onderzoek geworden is met veel nieuwe papers die gepubliceerd worden over PDSVRP en drone delivery. Er is dan ook nog zeer veel interessant onderzoek dat nog steeds gedaan kan worden in deze nieuwe boeiende tak van vehicle

routing problemen. Om het met de woorden van Amazon te zeggen "It would be easy to say, the sky is the limit, but that's not exactly true anymore, is it?".

Bibliografie

[1] Yogesh Agarwal, Kamlesh Mathur, and Harvey M Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989.

[2] Niels Agatz, Paul Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, 52(4):965–981, 2018.

[3] Amazon. Amazon prime air. `https://www.amazon.com/Amazon-Prime-Air`. Online; accessed 28 January 2022.

[4] Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Computers & operations research*, 107:32–42, 2019.

[5] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] Dimitris Bertsimas, Garrett Van Ryzin, et al. The dynamic traveling repairman problem. 1989.

[8] Yong Sik Chang and Hyun Jung Lee. Optimal delivery routing with wider drone-delivery areas along a shorter truck-route. *Expert Systems with Applications*, 104:307–317, 2018.

[9] I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996.

[10] Chun Cheng, Yossiri Adulyasak, and Louis-Martin Rousseau. Drone routing with energy function: Formulation and exact algorithm. *Transportation Research Part B: Methodological*, 139:364–387, 2020.

[11] André Alves Ferreira Sousa Conceição. Logistics challenges in a new distribution paradigm: Drone delivery. *Connect Robotics Case Study*, 2019.

[12] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[13] Rami Daknama and Elisabeth Kraus. Vehicle routing with drones. *arXiv preprint arXiv:1705.06431*, 2017.

[14] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[15] Júlia Cária de Freitas and Puca Huachi Vaz Penna. A variable neighborhood search for flying sidekick traveling salesman problem. *International Transactions in Operational Research*, 27(1):267–290, 2020.

[16] Mauro Dell'Amico, Michele Monaci, Corrado Pagani, and Daniele Vigo. Heuristic approaches for the fleet size and mix vehicle routing problem with time windows. *Transportation Science*, 41(4):516–526, 2007.

[17] Mauro Dell'Amico, Roberto Montemanni, and Stefano Novellani. Matheuristic algorithms for the parallel drone scheduling traveling salesman problem. *Annals of Operations Research*, 289(2):211–226, 2020.

[18] Luigi Di Puglia Pugliese and Francesca Guerriero. Last-mile deliveries by using drones and classical vehicles. In *International Conference on Optimization and Decision Science*, pages 557–565. Springer, 2017.

[19] Kevin Dorling, Jordan Heinrichs, Geoffrey G Messier, and Sebastian Magierowski. Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1):70–85, 2016.

[20] Jalel Euchi and Abdeljawed Sadok. Hybrid genetic-sweep algorithm to solve the vehicle routing problem with drones. *Physical Communication*, 44:101236, 2021.

[21] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation science*, 39(2):188–205, 2005.

[22] Sergio Mourelo Ferrandez, Timothy Harbison, Troy Weber, Robert Sturges, and Robert Rich. Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management (JIEM)*, 9(2):374–388, 2016.

[23] Michel Gendreau, Gilbert Laporte, Christophe Musaraganyi, and Éric D Taillard. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 26(12):1153–1173, 1999.

[24] Bruce Golden, Arjang Assad, Larry Levy, and Filip Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

[25] Bruce L Golden, Edward A Wasil, James P Kelly, and I-Ming Chao. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In *Fleet management and logistics*, pages 33–56. Springer, 1998.

[26] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[27] Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 86:597–621, 2018.

[28] Andy M Ham. Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming. *Transportation Research Part C: Emerging Technologies*, 91:1–14, 2018.

[29] Kückelhaus Markus Heutger Matthias. Unmanned aerial vehicles in logistics. *DHL trends*, 2020.

[30] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Diversion issues in real-time vehicle dispatching. *Transportation science*, 34(4):426–438, 2000.

[31] Martin Joerss, Jürgen Schröder, Florian Neuhaus, Christoph Klink, and Florian Mann. Parcel delivery: The future of last mile. *McKinsey & Company*, pages 1–32, 2016.

[32] E. kim. The most staggering part about amazon's upcoming drone delivery service. https://www.businessinsider.com/cost-savings-from-amazon-drone-deliveries-2016-6?international=true&r=US&IR=T, 2016. Online; accessed 16 April 2022.

[33] Sungwoo Kim and Ilkyeong Moon. Traveling salesman problem with a drone station. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):42–52, 2018.

[34] Patchara Kitjacharoenchai, Mario Ventresca, Mohammad Moshref-Javadi, Seokcheon Lee, Jose MA Tanchoco, and Patrick A Brunese. Multiple traveling salesman problem with drones: Mathematical model and heuristic approach. *Computers & Industrial Engineering*, 129:14–30, 2019.

[35] Doordash Labs. Building automation and robotics solutions to shape the future of last mile logistics. https://labs.doordash.com/. Online; accessed 28 January 2022.

[36] Gilbert Laporte and Silvano Martello. The selective travelling salesman problem. *Discrete applied mathematics*, 26(2-3):193–207, 1990.

[37] Allan Larsen. *The dynamic vehicle routing problem.* PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, 2000.

[38] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[39] David Levy, Kaarthik Sundar, and Sivakumar Rathinam. Heuristics for routing heterogeneous unmanned vehicles with fuel constraints. *Mathematical Problems in Engineering*, 2014, 2014.

[40] Feiyue Li, Bruce Golden, and Edward Wasil. A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 34(9):2734–2742, 2007.

[41] Apex Insight Ltd. Global parcel delivery market insight report 2021. *Research And Markets*, 2021.

[42] Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[43] Mario Marinelli, Leonardo Caggiani, Michele Ottomanelli, and Mauro Dell'Orco. En route truck–drone parcel delivery for optimal vehicle routing strategies. *IET Intelligent Transport Systems*, 12(4):253–261, 2018.

[44] Neil Mathew, Stephen L Smith, and Steven L Waslander. Planning paths for package delivery in heterogeneous multirobot teams. *IEEE Transactions on Automation Science and Engineering*, 12(4):1298–1308, 2015.

[45] Michalis Mavrovouniotis and Shengxiang Yang. Ant algorithms with immigrants schemes for the dynamic vehicle routing problem. *Information Sciences*, 294:456–477, 2015.

[46] Raïssa G Mbiadou Saleu, Laurent Deroussi, Dominique Feillet, Nathalie Grangeon, and Alain Quilliot. An iterative two-step heuristic for the parallel drone scheduling traveling salesman problem. *Networks*, 72(4):459–474, 2018.

[47] Liu Min and Wu Cheng. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13(4):399–403, 1999.

[48] Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

[49] Chase C Murray and Ritwik Raj. The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. *Transportation Research Part C: Emerging Technologies*, 110:368–398, 2020.

[50] Minh Anh Nguyen, Giang Thi-Huong Dang, Minh Hoàng Hà, and Minh-Trien Pham. The min-cost parallel drone scheduling vehicle routing problem. *European Journal of Operational Research*, 299(3):910–930, 2022.

[51] Luiz S Ochi, Dalessandro S Vianna, LÚcia Drummond, and André O Victor. An evolutionary hybrid metaheuristic for solving the vehicle routing problem with heterogeneous fleet. In *European Conference on Genetic Programming*, pages 187–195. Springer, 1998.

[52] Stefan Poikonen and Bruce Golden. The mothership and drone routing problem. *INFORMS Journal on Computing*, 32(2):249–262, 2020.

[53] Stefan Poikonen and Bruce Golden. Multi-visit drone routing problem. *Computers & Operations Research*, 113:104802, 2020.

[54] Andrea Ponza. Optimization of drone-assisted parcel delivery. 2016.

[55] Ritwik Raj, Dowon Lee, Seunghan Lee, Jose Walteros, and Chase Murray. A branch-and-price approach for the parallel drone scheduling vehicle routing problem. *Available at SSRN 3879710*, 2021.

[56] Gerhard Reinelt. Tsplib. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`. Online; accessed 1 May 2022.

[57] David Sacramento, David Pisinger, and Stefan Ropke. An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. *Transportation Research Part C: Emerging Technologies*, 102:289–315, 2019.

[58] Raïssa G Mbiadou Saleu, Laurent Deroussi, Dominique Feillet, Nathalie Grangeon, and Alain Quilliot. The parallel drone scheduling problem with multiple drones and vehicles. *European Journal of Operational Research*, 300(2):571–589, 2022.

[59] Daniel Schermer, Mahdi Moeini, and Oliver Wendt. Algorithms for solving the vehicle routing problem with drones. In *Asian Conference on Intelligent Information and Database Systems*, pages 352–361. Springer, 2018.

[60] Daniel Schermer, Mahdi Moeini, and Oliver Wendt. A b ranch-and-cut approach and alternative formulations for the traveling salesman problem with drone. *Networks*, 76(2):164–186, 2020.

[61] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[62] Marlin W. Ulmer and Barrett W. Thomas. Same-day delivery with heterogeneous fleets of drones and vehicles. *Networks*, 72(4):475–505, 2018.

[63] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[64] Thibaut Vidal. Split algorithm in o (n) for the capacitated vehicle routing problem. *Computers & Operations Research*, 69:40–47, 2016.

[65] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[66] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[67] Desheng Wang, Peng Hu, Jingxuan Du, Pan Zhou, Tianping Deng, and Menglan Hu. Routing and scheduling for hybrid truck-drone collaborative parcel delivery with independent and truck-carried drones. *IEEE Internet of Things Journal*, 6(6):10483–10495, 2019.

[68] Xingyin Wang, Stefan Poikonen, and Bruce Golden. The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, 11(4):679–697, 2017.

[69] Zheng Wang and Jiuh-Biing Sheu. Vehicle routing problem with drones. *Transportation research part B: methodological*, 122:350–364, 2019.

[70] Google Wing. How it works. `https://wing.com/how-it-works/`. Online; accessed 29 November 2021.

[71] Ivan Xavier. Cvrplib. `http://vrp.atd-lab.inf.puc-rio.br`. Online; accessed 26 March 2022.

[72] Xin Xiao. A direct proof of the 4/3 bound of lpt scheduling rule. In *Proceedings of the 2017 5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017)*, pages 486–489. Atlantis Press, 2017/04.

[73] Jingyang Xu and Rakesh Nagi. Identical parallel machine scheduling to minimise makespan and total weighted completion time: a column generation approach. *International Journal of Production Research*, 51(23-24):7091–7104, 2013.

[74] Emine Es Yurek and H Cenk Ozmutlu. A decomposition-based iterative optimization algorithm for traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 91:249–262, 2018.