applied sciences

MDPI

Article

Combining Parallel Computing and Biased Randomization for Solving the Team Orienteering Problem in Real-Time

Javier Panadero [1], Majsa Ammouriova [1], Angel A. Juan [2,*], Alba Agustin [3], Maria Nogal [4] and Carles Serrat [5]

1 IN3—Computer Science Department, Universitat Oberta de Catalunya, 08018 Barcelona, Spain; jpanaderom@uoc.edu (J.P.); mammouriova@uoc.edu (M.A.)
2 Department of Applied Statistics and Operations Research, Universitat Politècnica de València, 03801 Alcoy, Spain
3 Statistics, Informatics & Mathematics Department, Public University of Navarra, 31006 Pamplona, Spain; albamaria.agustin@unavarra.es
4 Materials, Mechanics, Management & Design Department, Delft University of Technology, 2628 CN Delft, The Netherlands; M.Nogal@tudelft.nl
5 Department of Mathematics, Technical University of Catalonia—BarcelonaTech, 08028 Barcelona, Spain; carles.serrat@upc.edu
* Correspondence: ajuanp@uoc.edu

Abstract: In smart cities, unmanned aerial vehicles and self-driving vehicles are gaining increased concern. These vehicles might utilize ultra-reliable telecommunication systems, Internet-based technologies, and navigation satellite services to locate their customers and other team vehicles to plan their routes. Furthermore, the team of vehicles should serve their customers by specified due date efficiently. Coordination between the vehicles might be needed to be accomplished in real-time in exceptional cases, such as after a traffic accident or extreme weather conditions. This paper presents the planning of vehicle routes as a team orienteering problem. In addition, an 'agile' optimization algorithm is presented to plan these routes for drones and other autonomous vehicles. This algorithm combines an extremely fast biased-randomized heuristic and a parallel computing approach.

Keywords: team orienteering problem; real-life optimization; parallel computing; biased randomization; smart cities; unmanned aerial vehicles

Citation: Panadero, J.; Ammouriova, M.; Juan, A.A.; Agustin, A.; Nogal, M.; Serrat, C. Combining Parallel Computing and Biased Randomization for Solving the Team Orienteering Problem in Real-Time. *Appl. Sci.* **2021**, *11*, 12092. https://doi.org/10.3390/app112412092

Academic Editor: Bernabe Dorronsoro

Received: 16 November 2021
Accepted: 16 December 2021
Published: 19 December 2021

1. Introduction

Sustainable cities and communities are identified as one of the 17 sustainable-development goals proposed by the UN [1]. The achievement of this goal requires the significant renewal of the way urban space is perceived. In this context, smart cities are envisaged as the main driver of such a transformation. Advanced materials, sensors, electronics, and networks embedded in our physical and social systems constitute the core concept of a smart city, allowing for more sustainable and resilient societies [2].

Smart cities are an ever-changing environment [3], which aims to provide high-quality life to its citizens supported by the advancement in information and communication technology and the integration of the Internet of Things (IoT). As a result, new mobility modes are considered such as ride-sharing [4] or the incorporation of electric vehicles [5].

Unmanned aerial vehicles (UAVs), commonly known as drones, have attracted significant attention in the last decade, given their potential for new applications [6]. For example, in the case of smart cities, they can work as aerial base stations, either collecting data from mobile and ground sensors or serving as sensor-mounted aerial platforms [7]. In that way, an on-demand data service can be realized, covering a larger number of sensors and users.

A major disadvantage of these devices is their limited energy capacity, which adds a certain complexity to implementing these types of services, especially when serving remote sites [8]. Several operational research lines arise to cope with this limitation: (*i*) the

optimal drone placement (ODP) problem, which guarantees the coverage of static or dynamic targets minimizing the required energy [9,10]; (*ii*) the problem of selecting the optimum charging station once the drones have finished their tasks [11]; and (*iii*) the route-planning problem and all its variants [12,13]. Considering that a fleet of drones can work in a coordinated manner to achieve a common goal, such as the aerial surveillance of an extended area, the problem of coordinating their individual paths accounting for their available energy can be envisioned as a team orienteering problem (TOP). The objective of the TOP would be, following the case of the aerial surveillance, to maximize the area covered by the fleet of drones. Therefore, the efficiency of the fleet of drones is maximized. The drone scheduling problem (DSP) is an extension of the TOP that considers multiple depots (stations) [14]. Figure 1 depicts the conceptual framework of several operational research problems regarding drones.

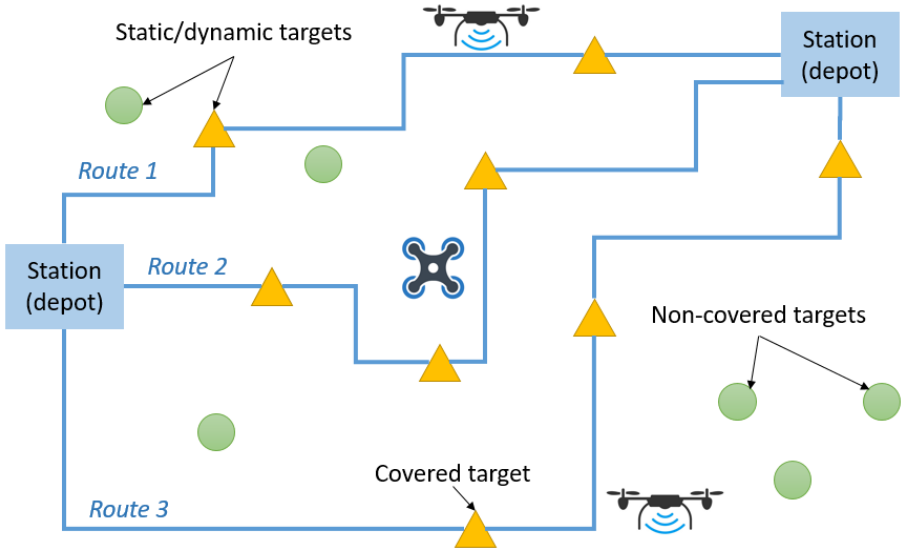

Figure 1. Conceptual definition of some Operational Research problems regarding drones.

Smart cities allow for real-time data, nevertheless, their potential can come to fruition only if combined with real-time decision-making. This is not only the case for drones, where optimal routing should be established accounting for the dynamic conditions (e.g., dynamic targets) but also for other types of autonomous vehicles working collaboratively, which might be increasingly frequent in transportation and mobility activities. Traditional optimization approaches cannot handle real-time conditions effectively [15]. Therefore, new optimization approaches are required to deal with large-scale systems in 'real-time' (e.g., less than one second). The development of efficient solving approaches becomes incredibly challenging given the large-scale problems that real applications involve, and the fact that the TOP is NP-hard in nature like the most of routing problems [16].

In order to tackle the challenges imposed by smart cities, this paper proposes the use of *agile optimization* algorithms [17] to solve TOPs with dynamic conditions in 'real-time', i.e., below one second of wall-clock time. In this context, optimization algorithms are re-designed to become (*i*) extremely fast to support real-time decision-making; (*ii*) extendable to adapt parallelization techniques; (*iii*) flexible to handle different problems; (*iv*) parameter loss to avoid parameter fine-tuning; and (*v*) dynamic to rerun as new data become available (re-optimization).

Agile optimization includes the hybridization of biased-randomized algorithms [18] and parallel computing [19]. The biased randomized algorithms result from utilizing skewed probability distributions in deterministic heuristics [20]. The deterministic heuristics handle even large-scale problems efficiently. The utilized distributions introduce a randomized common-sense effect in these heuristics and result in probabilistic algorithms. The probabilistic algorithms could be run in parallel to generate different solutions using

an affordable computing device. Thus, thousands of these probabilistic algorithms could be run simultaneously. The probabilistic algorithms generate many alternative solutions in the same clock time compared to one solution found by the deterministic heuristic. Some of these solutions could outperform the solution determined by the deterministic heuristic. The biased-randomization techniques have been successfully tested on a variety of optimization problems [21–23].

The parallel algorithms were utilized by a number of researchers in solving routing problems. For example, Roberge and Tarbouchi [24] developed a framework to download data from sensors by unmanned aerial vehicles. In their research, they used a genetic algorithm as the single source algorithm to optimize the routes of the vehicles. The single algorithm was run in parallel on a graphics processing unit and aimed to avoid collision. Yelmewad and Talawar [25] used a graphics processing unit based parallel strategy to reduce execution time needed to solve the vehicle routing problem. Other researchers utilized multi-core approach for parallel computing. Abbasi et al. [26] aimed to reduce the cost of intelligent systems in transportation. For that purpose, they studied multi-core processors as well as graphics processing units and concluded that the parallelization resources are efficiently utilized. Multi-threading in a multi-core system was utilized in the multi-start approach [27]. Other researchers studied used protocol for data transmission between different nodes in a network, such as Huang et al. [28]. These protocols aim to reduce delays and energy consumption in different IoT devices.

In previous publications, the stochastic version of the TOP was investigated [29,30]. Simheuristic approach was used to handle the stochasticity of travel times. Monte Carlo simulation was integrated with a biased randomized heuristic in Panadero et al. [29], and Panadero et al. [30] combined a saving based heuristic and the variable neighborhood search metaheuristic and integrated them in a simheuristic approach. Furthermore, the dynamic change of customers' reward was investigated by Reyes-Rubiano et al. [31]. For this purpose, a biased randomized heuristic was extended into a learnheuristic. Therefore, the main achievements of this paper are (*i*) proposing a fast heuristic able to generate real-time solutions of reasonably good quality for the TOP; (*ii*) extending the heuristic into a biased-randomized algorithm to generate many alternative promising solutions—some of them might outperform the original one found by the heuristic; and (*iii*) integrating the biased-randomized algorithm into a parallelization framework to generate solutions in real-time. This paper presents a parallelized biased-randomized algorithm to solve the TOP.

The remaining of the paper is structured as follows. Section 2 provides some related work on the TOP, which serves as a scenario for testing our concepts, and Section 3 introduces a mathematical formulation of the considered problem. Section 4 presents a biased-randomized algorithm to be easily parallelized. The actual parallelization concepts are analyzed in Section 5. Section 6 describes the set of computational experiments that have been carried out to test both exact and approximation-based solution methods. Section 7 analyzes in detail the obtained results. Finally, Section 8 summarizes the main contributions of the work and possible further future work.

2. Related Work

The Orienteering Problem (OP) is among the most widely studied combinatorial optimization problems, and even the Traveling Salesman Problem (TSP) with profits or Selective TSP is commonly named as the OP [32]. The term OP was first introduced by Golden et al. [33] that defines the TSP involving Knapsack constraints. Therefore, the goal of the OP is to simultaneously (*i*) minimize the travel cost, which usually appears as a constraint, and (*ii*) maximize the collected profit associated with each node. Moreover, it is remarkable to mention that the OP is NP-hard [34]. We refer the reader to the surveys Feillet et al. [35] for further information about OP solution approaches and more recently in [36,37] for further information about OP numerous extensions and recent variants.

The Team Orienteering Problem (TOP) is a well-known extension of the OP [38], and it is also referred as the Vehicle Routing Problem (VRP) with profits [37]. In particular, the TOP considers a group of agents for collecting the profits and provides a solution with several tours [39] and, in that sense, the connotation of the TSP turns into the VRP. In addition, most of the TOP's studies originate from previous studies devoted to the OP and VRP that have been adapted or extended to the context of the TOP.

Regarding the TOP solution techniques, from last decades until now, there are few solution approaches based on exact methods such as Branch-and-Price [40,41], Branch-and-Cut [42], and Column Generation [43]. However, to tackle large instances and reduce computational times, we observe that (meta)heuristic methods are desirable. These methods might find (near)optimal solutions for the TOP problem, such as Tabu Search [38,44], Variable Neighborhood Search [45], Ant Colony [46], Memetic Algorithm [47], Simulated Annealing [48], Particle Swarm Optimization [49], Genetic Algorithm [50], Pareto Mimic Algorithm [51], and Hybrid Harmony Search [52]. In recent years, simheuristics have been introduced to solve large combinatorial optimization problems [53–55]. Regarding the TOP, we observe that all these techniques in the state of the art have been tested to prove that their solutions, in terms of objective function value and computational times, are competitive ones. Furthermore, we find an extensive comparison of algorithms in the literature on instances of Chao's benchmark [39], such as the comparison in Dang et al. [49]. Commonly, the emerging algorithms focus on reaching the best-known-solution (BKS) for these instances in the benchmark. However, in this paper, we constrain computational time to only one second. Then, we compare our obtained solutions with the BKS and show with a complete analysis that our solutions reach the BKS in most instances.

3. A Formal Description of the Problem

The TOP can be mathematically formalized as follows. Let us assume an undirected graph $G = (N, A)$, where: (*i*) $N = \{0, 1, 2, \ldots, n+1\}$ includes the set of nodes accounting for n customers, $N' = \{1, 2, \ldots, n\}$, a source node 0, and a sink node $n+1$; and (*ii*) A is the set of edges (i, j), with $i \neq j$, connecting nodes $i \in N \setminus \{n+1\}$ and $j \in N \setminus \{0\}$. At the source, node 0, the team is composed of a limited number of vehicles $m \geq 1$. Furthermore, there is a maximum time, $t_0 > 0$, for completing each open route (from its source node to the sink node, with at least one customer node in between). Each customer node $i \in N'$ has a constant reward $u_i > 0$, while $u_0 = u_{n+1} = 0$. Rewards can only be collected on the first visit to a node. Each edge $(i, j) \in A$ has an associated travel time, $t_{ij} > 0$. It is assumed that $t_{ij} = t_{ji}$, i.e., for $(i, j) \in A$, the matrix of travel times, $T = [t_{ij}]$, is a symmetric one that satisfies the triangle inequality.

A solution to the TOP is a set of feasible routes departing from the source node, visiting a subset of customers in a specified order, and arriving at the sink node. In other words, each route starts at the source node 0, collects the reward from one or more customers in N', and ends at the sink node, without exceeding the maximum travel time allowed, t_0. Let us consider the binary decision variable x_{ij}, which takes the value 1 if the edge $(i, j) \in A$ is used by a vehicle to collect the reward at node j, and 0 otherwise. In addition, let us define the continuous variable w_i, which represents the total travel time that the vehicle has spent after visiting customer i. The objective function is given by the maximization of the total collected reward:

$$\max \sum_{(i,j) \in A} u_i x_{ij}. \tag{1}$$

This objective function is subject to the following constraints:

$$\sum_{j \in N \setminus \{0\}} x_{ij} \leq 1, \quad \forall i \in N', \tag{2}$$

$$\sum_{i \in N \setminus \{n+1\}} x_{ij} \leq 1, \quad \forall j \in N', \tag{3}$$

$$\sum_{k \in N \setminus \{n+1\}} x_{ki} = \sum_{j \in N \setminus \{0\}} x_{ij}, \quad \forall i \in N', \tag{4}$$

$$\sum_{j \in N'} x_{0j} = \sum_{i \in N'} x_{i\,(n+1)}, \tag{5}$$

$$\sum_{j \in N'} x_{0j} \leq m, \tag{6}$$

$$w_j \leq t_0, \quad \forall j \in N \setminus \{0\}, \tag{7}$$

$$w_i + t_{ij}\, x_{ij} - w_j \leq t_0 - t_0\, x_{ij}, \quad \forall i \in N \setminus \{n+1\}, \forall j \in N \setminus \{0, i\}, \tag{8}$$

$$\sum_{i \in N} x_{ii} + x_{0\,(n+1)} = 0, \tag{9}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A, \tag{10}$$

$$w_i \geq 0, \quad \forall i \in N \setminus \{0\}. \tag{11}$$

Constraints (2) and (3) impose that each customer node has at most one edge departing from it or entering it, respectively. In addition, constraint (4) imposes that, for each customer node, the number of incoming edges is equal to the number of outgoing edges (due to the previous constraints, this value will be either 0 or 1). Constraint (5) ensures that the number of routes starting at the source node is the same as the number of routes arriving at the sink node. Constraint (6) forces that the number of routes must be less or equal to the number of available vehicles m. Two constraints, (7) and (8), are introduced for both the connectivity of the solution and the maximum travel time requirement. Constraint (9) avoids degenerated routes. Finally, Constraints (10) and (11) define the range of the associated variables.

4. From a Heuristic to a Biased-Randomized Algorithm

In this paper, a heuristic is extended to a biased-randomized algorithm to solve the TOP. The proposed algorithm extends the concept of 'savings' introduced by Clarke and Wright [56] to the characteristics of the TOP: (*i*) different origin and destination depots, (*ii*) some of the customers might not be covered, and (*iii*) the reward as well as the savings in time or distance are considered to construct routes. Then, the constructive heuristic is extended to utilize skewed probability distributions to introduce a non-uniform randomization effect into the procedure of constructing solutions.

Figure 2 presents the proposed constructive heuristic. An initial dummy solution is built for each customer $i \in N'$; a vehicle departs from the origin depot (node 0), visits customer i, and then head towards the destination depot (node $n+1$). If a route associated with customer i in the dummy solution does not satisfy the driving-range constraint, customer i is discarded from the solution because it cannot be served with the current settings of the problem.

Next, the 'enriched savings' associated with each edge connecting two different customers are computed; thus, the benefits obtained by visiting both customers in the same route instead of using two different routes are calculated. The enriched savings of an edge considers the travel time required to traverse the edge and the aggregated reward generated by visiting both customers. Therefore, the enriched savings associated with an edge (i, j), s'_{ij}, are defined as $s'_{ij} = \alpha \cdot s_{ij} + (1 - \alpha) \cdot (u_i + u_j)$ to account for the trade-off between time-based savings $s_{ij} = t_{i(n+1)} + t_{0j} - t_{ij}$, and the aggregated reward, $u_i + u_j$. Here, $\alpha \in (0, 1)$ is a parameter that depends on the rewards of customers. Experiments are used to determine the value of α. In general, α is set close to zero in problems with high heterogeneity rewards and one in problems with homogeneous customer rewards.

For each edge, we obtain its unique associated savings; recall that the matrix of travel times is symmetric, so the savings do not depend on the direction in which the edge is traversed. These savings are associated with arcs that connect two potential routes to be merged. Then, the list of arcs can be sorted from higher to lower savings. Routes are merged based on the sorted savings list. In each iteration of the merging process, the arc at

the top of the sorted list is selected, and the two routes connected by this arc are merged into a new route if the driving-range constraint is not violated. Finally, the list of routes is decreasingly sorted according to the total reward. Routes with the highest rewards are selected, and the number of selected routes equals the size of the vehicle fleet.

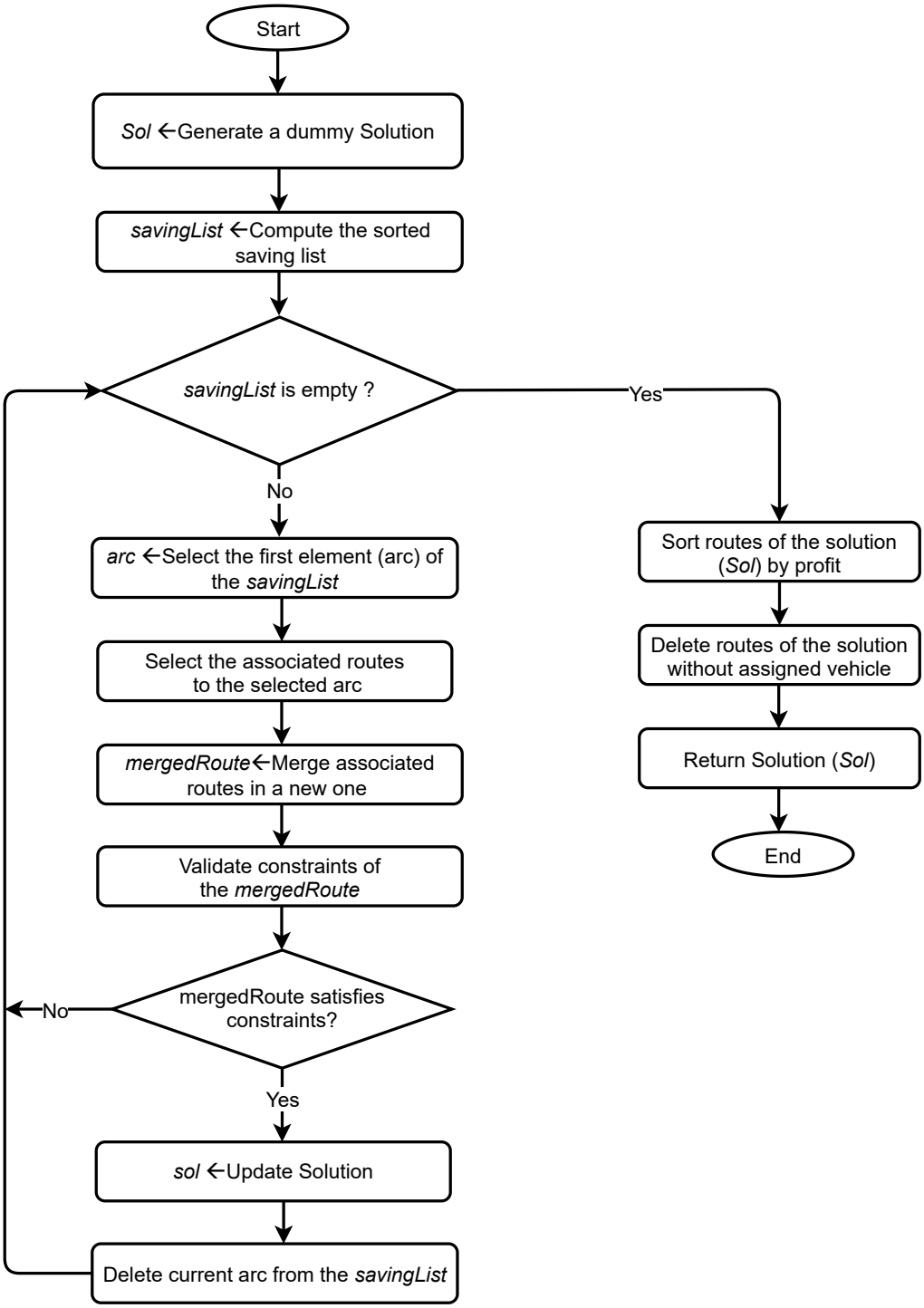

Figure 2. Flowchart of the Heuristic approach.

The described heuristic is extended into a probabilistic algorithm as follows: The greedy behavior of the heuristic is altered by combining it with biased-randomization techniques to introduce non-uniform random behavior [57]. In our work, we utilized geometric probability distribution with a parameter β ($0 < \beta < 1$). Parameter β of the geometric distribution controls the greediness of the randomized behavior. In our

experiments, we set the value of β to 0.3. This value was a result of parameter tuning experiments. Thus, the heuristic becomes randomized algorithm and utilizes the greedy behavior of the original heuristic.

5. Extending to a Parallel Biased-Randomized Algorithm

The intrinsic characteristics of biased-randomized algorithms make them a good candidate to be parallelized. They could be utilized in a multi-start framework, which is a sequential and iterative approach. In each iteration of this approach, a different solution is generated [58]. In our paper, multi-start methods are composed of two phases: In the first one, a new solution is generated using the biased-randomized heuristic, and in the second one, the algorithm compares the newly generated solution with the best solution obtained so far to update the latter whenever appropriate.

Instead of using this sequential approach, multiple instances of the biased-randomized algorithm can be run in parallel (e.g., each using a different computer thread or core) by assigning a different seed of the pseudo-random number generator to each instance (Figure 3). In parallel programming, these types of methods—where the same code is executed using different input parameters without requiring communication or dependency between the processes—are known as *embarrassingly parallel algorithms* [59]. These algorithms are usually easy to adapt for parallel execution, and they are good candidates to be executed using massively parallel processing architectures [60].

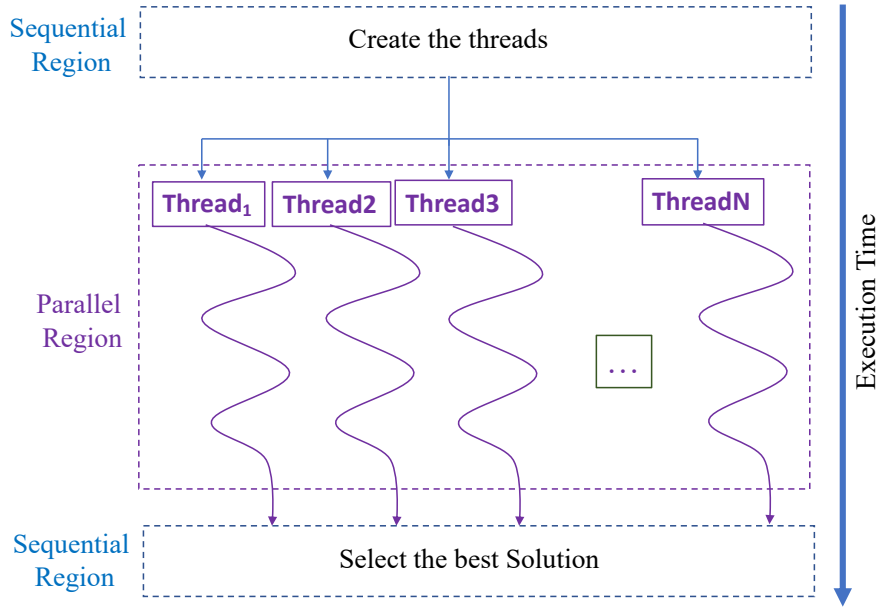

Figure 3. Executing different instances of biased-randomized algorithm in a parallel way.

In the context of unmanned aerial vehicles or self-driving vehicles, decisions should be made in short times, e.g., seconds or even in milliseconds. Using a distributed computing architecture—such as cloud platforms—is an unsuitable approach in this context due to the relatively high latencies of the network. Thus, high-performance computing architectures on-chip must be employed. These architectures could be multi-core processors or graphics processing units (GPUs). These processor units consist of hundreds of smaller low-energy cores that are divided into groups, streaming multiprocessors [61]. The architecture enables parallel computation with relative energy efficiency compared to traditional processors. Using modern multi-core processors represents another appropriate approach for running embarrassingly parallel algorithms. Multi-core processors have become very popular in the last years, in part because modern computers contain several processing units or cores in a single chip [62]. Moreover, multi-core processors can execute several threads by core simultaneously (hyper-threading), turning these processors into a valuable and inexpensive

approach to execute parallel programs. The union of biased-randomized algorithms and parallel architectures makes a perfect combination to obtain high-quality solutions in complex systems scenarios in the order of just a few seconds or even milliseconds.

In our approach (Algorithm 1), we have used a Multi-core shared memory programming paradigm. Thus, the algorithm executes asynchronous threads that contain as input parameters an independent instance of the BR-heuristic, a seed (pseudo-random number), and a list (*solutionList*). The list is a shared variable by all the threads, and each thread saves the best-found solution in one second using its instance of the BR-Algorithm in this list. After creating and running all the threads, the algorithm waits until all threads are executed. Notice that the algorithm ends its parallel execution at this point, and the shared variable *solutionList* contains all the solutions found by all the threads. Then, the program executes its last step sorting the list of solutions by reward and selecting the best solution (*bestSolution*).

Algorithm 1 Parallel Biased-Randomized Algorithm (numberOfThreads).

```
1: solutionList ← ∅
2: for i ← 1 to numberOfThreads do
3:     seed ← PseudoRandomNumberGenerator()
4:     t ← thread(i, BRTopAlgorithm, seed, solutionList) %Create a thread
5:     t.start() %Thread starts to execute
6: end for
7: SynchronizeThreads() %Programm waits until all threads finish
8: sortedSolutions ← SortSolutionsByReward(solutionList)
9: bestSolution ← selectBestSolution(sortedSolutions)
10: return bestSolution
```

6. Numerical Experiments

This section describes the numerical experiments we have carried out to test the concept of agile optimization when applied to the TOP. First, we have employed exact methods to solve the optimization problem. This initial experiment allows us to understand the possible limitations of these methods in terms of computational times and the size of the instances they can solve in practical applications. Then, we have employed our parallelized biased-randomized algorithm to generate solutions in real-time. Finally, these solutions have been compared with the optimal or near-optimal ones available in the scientific literature whenever possible.

Both solving approaches have been tested using the benchmark instances presented in [39], which are available in the repository http://www.mech.kuleuven.be/en/cib/op/instances (accessed on 17 December 2021). This benchmark has been widely used in previous works to test the performance of algorithms aimed at solving the deterministic TOP. The benchmark comprises a total of 354 instances, which are divided into seven different sets (Table 1). The number of nodes, the node locations, and the rewards are identical for all instances within a set. However, the maximum allowed driving range (t_0) and the number of vehicles vary between the instances. The sum of all customers' rewards is shown in column "Rewards" in Table 1. The number of vehicles varies between 2 and 4, and the driving range varies between the values tabulated in Table 1. Therefore, from one instance to another, the driving range is increased by the tabulated step. Each instance in a set has a nomenclature *px.y.z*, where *x* denotes the set number, *y* is the number of vehicles, and *z* symbolizes the maximum driving range.

Table 1. Characteristics of the benchmark sets used in the experiments.

Set (x)	Nodes	Reward	Vehicles (y)	Driving Range (z)	Number of Instances
p1	32	285	2 3 4	2.5–42.5 1.7–28.3 1.2–21.2	56
p2	21	450	2 3 4	7.5–22.5 5.0–15.0 3.8–11.2	33
p3	33	800	2 3 4	7.5–55 5.0–36.7 3.8–27.5	60
p4	100	1306	2 3 4	25.0–120.0 16.7–80.0 12.5–60.0	60
p5	66	1680	2 3 4	2.5–65.0 1.7–43.3 1.2–32.5	78
p6	64	1344	2 3 4	7.5–40.0 5.0–26.7 3.8–20.0	42
p7	102	1458	2 3 4	10.0–200.0 6.7–133.3 5.0–100.0	60

6.1. Solving the TOP Using Exact Methods

The proposed mathematical model has been implemented using the IBM ILOG CPLEX Optimization Studio v12.6.2, and the computations were conducted on a PC with an Intel i5-6500 quad-core processor running at 3.6 GHz and with 20 GB of RAM. The CPLEX default options have been used in the experimentation, with the specification of the following settings: (*i*) a time limit of 3600 s, (*ii*) a relative gap of 0.01%, and (*iii*) a tolerance integrality of 0.001%. The computational results are reported in Table 2. Columns 1 and 5 identify the instance for sets $p1$ and $p2$, respectively. Columns 2 and 6 show the solution value obtained with CPLEX. The computational time (in seconds) requested to obtain these values is given in columns 3 and 7. Furthermore, the CPLEX final status indicates whether the solution is guaranteed to be optimal or not. In the latter case, the best-known solution (BKS) from the literature is provided.

It is observed that in set $p2$ (instances with 21 nodes), optimal solutions can usually be achieved after a few seconds or minutes. However, in set $p1$ (instances with 33 nodes), only a few optimal solutions can be achieved, whereas others cannot reach the optimal solution even when employing up to 60 min of computation. It is noticed that some solutions differ significantly from the BKS, such as instances $p1.2.r$ and $p1.3.p$. In conclusion, exact methods are not a feasible option to provide real-time solutions for large-sized instances with hundreds of nodes.

Table 2. Computational results for the *p*1 and *p*2 instances in CPLEX.

Instances	Solution	Time (s)	BKS Found	Instances	Solution	Time (s)	BKS Found
p1.2.b	15	0.06	YES	p2.2.a	90	0.39	YES
p1.2.c	20	0.47	YES	p2.2.b	120	7.13	YES
p1.2.d	30	17.46	YES	p2.2.c	140	74.96	YES
p1.2.e	45	1372.59	YES	p2.2.d	160	209.65	YES
p1.2.f	80	3600.01	YES	p2.2.e	190	541.92	YES
p1.2.g	80	3600.01	NO (BKS 90)	p2.2.f	200	17.58	YES
p1.2.h	100	3600.01	NO (BKS 110)	p2.2.g	200	136.81	YES
p1.2.i	105	3600.01	NO (BKS 135)	p2.2.h	230	1016.16	YES
p1.2.j	135	3600.01	NO (BKS 155)	p2.2.i	230	3600.01	YES
p1.2.k	150	3600.01	NO (BKS 175)	p2.2.j	230	3600.01	NO (BKS 260)
p1.2.l	180	3600.01	NO (BKS 195)	p2.2.k	270	3600.01	NO (BKS 275)
p1.2.m	175	3600.01	NO (BKS 215)	p2.3.a	70	0.03	YES
p1.2.n	160	3600.01	NO (BKS 235)	p2.3.b	70	0.22	YES
p1.2.o	215	3600.01	NO (BKS 240)	p2.3.c	105	0.34	YES
p1.2.p	215	3600.01	NO (BKS 250)	p2.3.d	105	0.47	YES
p1.2.q	230	3600.01	NO (BKS 265)	p2.3.e	120	2.06	YES
p1.2.r	205	3600.01	NO (BKS 280)	p2.3.f	120	6.36	YES
p1.3.c	15	0.26	YES	p2.3.g	145	12.2	YES
p1.3.d	15	4.85	YES	p2.3.h	165	84.39	YES
p1.3.e	30	7.75	YES	p2.3.i	200	11.71	YES
p1.3.f	40	46.75	YES	p2.3.j	200	8.01	YES
p1.3.g	50	2818.14	YES	p2.3.k	200	23.77	YES
p1.3.h	70	3600.01	YES	p2.4.a	10	0.02	YES
p1.3.i	105	3600.01	YES	p2.4.b	70	0.02	YES
p1.3.j	105	3600.01	NO (BKS 115)	p2.4.c	70	0.16	YES
p1.3.k	115	3600.01	NO (BKS 135)	p2.4.d	70	0.14	YES
p1.3.l	125	3600.01	NO (BKS 155)	p2.4.e	70	0.17	YES
p1.3.m	165	3600.01	NO (BKS 175)	p2.4.f	105	0.28	YES
p1.3.n	165	3600.01	NO (BKS 190)	p2.4.g	105	0.55	YES
p1.3.o	165	3600.01	NO (BKS 205)	p2.4.h	120	1.34	YES
p1.3.p	175	3600.01	NO (BKS 220)	p2.4.i	120	3.21	YES
p1.3.q	220	3600.01	NO (BKS 230)	p2.4.j	120	5.89	YES
p1.3.r	210	3600.01	NO (BKS 250)	p2.4.k	180	17.61	YES
p1.4.d	15	0.95	YES	-	-	-	-
p1.4.e	15	3.88	YES	-	-	-	-
p1.4.f	25	7.75	YES	-	-	-	-
p1.4.g	35	11.54	YES	-	-	-	-
p1.4.h	45	165.71	YES	-	-	-	-
p1.4.i	60	759.26	YES	-	-	-	-
p1.4.j	75	2914.08	YES	-	-	-	-
p1.4.k	100	3600.01	YES	-	-	-	-
p1.4.l	120	3600.01	YES	-	-	-	-
p1.4.m	125	3600.01	NO (BKS 130)	-	-	-	-
p1.4.n	130	3600.01	NO (BKS 155)	-	-	-	-
p1.4.o	155	3600.01	NO (BKS 165)	-	-	-	-
p1.4.p	155	3600.01	NO (BKS 175)	-	-	-	-
p1.4.q	165	3600.01	NO (BKS 190)	-	-	-	-
p1.4.r	165	3600.01	NO (BKS 210)	-	-	-	-

6.2. Solving the TOP with Our Agile Optimization Algorithm

The proposed heuristic has been implemented using Java SE 8.0 and tested on a workstation with a multi-core processor Intel Xeon E5-2650 v4 with 16 cores and 32 GB of RAM. Each instance is run during 1 s using a different number of threads (from 1 to 128) to

evaluate the quality of the solution as the number of threads increases. Each thread was a different algorithm run, and each run used a different seed for the pseudo-random number generator. In this way, each thread explores a different path in the solution space—but always keeping the logic behind the constructive heuristic. Increasing the number of threads increases the total computing time (as more computing resources are running in parallel) but not the wall-clock time, which is still 1 s. To assess the effectiveness of the proposed approach, the obtained solutions have been compared against the BKS from the literature. In particular, the performance of the algorithm is measured using the current BKS reported by [51]. Tables 3 and 4 show instances and their found solutions. The BKS is tabulated as the obtained reward and the computational time—in seconds—to reach it. Our best solution (OBS) for a specific number of threads is tabulated. The computational time using 128 threads is only recorded in Tables 3 and 4. We have used this time because 128 threads found the best OBS among the different number of threads. The average percentage gap between the OBS for 128 threads and the BKS is calculated. Notice that increasing the number of used threads might find better solutions. For example, the solution found using 128 threads is better than the one found using 16 threads for instance p1.3.m (Table 3). These results are discussed in the next section.

Table 3. Results for set *p*1.

Instance	BKS ([51])		Our Best Solution (OBS)									Gap (%) [a] — [b]
	Reward [a]	Time (s)	Thr_1	Thr_2	Thr_4	Thr_8	Thr_{16}	Thr_{32}	Thr_{64}	Thr_{128} [b]	Time (s)	
p1.2.b	15	3.00	15	15	15	15	15	15	15	15	0.09	0.00
p1.2.c	20	2.00	20	20	20	20	20	20	20	20	0.19	0.00
p1.2.d	30	4.30	30	30	30	30	30	30	30	30	0.74	0.00
p1.2.e	45	3.40	45	45	45	45	45	45	45	45	0.49	0.00
p1.2.f	80	4.90	80	80	80	80	80	80	80	80	0.04	0.00
p1.2.g	90	3.00	85	85	85	85	85	85	85	90	0.62	0.00
p1.2.h	110	5.10	105	105	105	105	105	105	105	110	0.25	0.00
p1.2.i	135	12.10	130	130	130	130	130	130	130	135	0.44	0.00
p1.2.j	155	8.30	150	150	150	150	150	150	155	155	0.98	0.00
p1.2.k	175	31.00	170	170	175	175	175	175	175	175	0.24	0.00
p1.2.l	195	2.70	185	185	185	185	185	190	190	195	0.29	0.00
p1.2.m	215	45.30	200	205	205	205	205	205	205	215	0.56	0.00
p1.2.n	235	6.40	230	230	230	230	230	230	230	235	0.58	0.00
p1.2.o	240	10.30	235	235	235	235	235	235	235	235	0.27	2.13
p1.2.p	250	10.10	240	240	240	240	240	240	240	245	0.61	2.04
p1.2.q	265	7.80	250	250	250	250	255	255	255	255	0.33	3.92
p1.2.r	280	9.20	275	275	275	275	275	275	275	275	0.43	1.82
p1.3.c	15	3.00	15	15	15	15	15	15	15	15	0.73	0.00
p1.3.d	15	3.00	15	15	15	15	15	15	15	15	0.60	0.00
p1.3.e	30	2.00	30	30	30	30	30	30	30	30	0.88	0.00
p1.3.f	40	2.00	40	40	40	40	40	40	40	40	0.36	0.00
p1.3.g	50	4.30	50	50	50	50	50	50	50	50	0.51	0.00
p1.3.h	70	4.60	70	70	70	70	70	70	70	70	0.16	0.00
p1.3.i	105	2.00	95	95	95	95	100	100	100	105	0.50	0.00
p1.3.j	115	16.90	110	110	110	115	115	115	115	115	0.09	0.00
p1.3.k	135	5.80	125	125	130	130	130	130	130	130	0.53	3.85
p1.3.l	155	6.50	155	155	155	155	155	155	155	155	0.49	0.00
p1.3.m	175	19.40	160	160	160	160	165	165	165	170	0.37	2.94
p1.3.n	190	6.70	185	185	190	190	190	190	190	190	0.21	0.00
p1.3.o	205	8.30	205	205	205	205	205	205	205	205	0.76	0.00

Table 3. *Cont.*

Instance	BKS ([51])		Our Best Solution (OBS)									Gap (%)
	Reward [a]	Time (s)	Thr_1	Thr_2	Thr_4	Thr_8	Thr_{16}	Thr_{32}	Thr_{64}	Thr_{128} [b]	Time (s)	[a] — [b]
p1.3.p	220	6.60	215	215	215	215	215	215	215	215	0.91	2.33
p1.3.q	230	9.10	225	230	230	230	230	230	230	230	0.23	0.00
p1.3.r	250	13.70	240	240	240	240	240	240	240	240	0.34	4.17
p1.4.d	15	3.00	15	15	15	15	15	15	15	15	0.73	0.00
p1.4.e	15	3.00	15	15	15	15	15	15	15	15	0.99	0.00
p1.4.f	25	3.00	25	25	25	25	25	25	25	25	0.99	0.00
p1.4.g	35	2.00	35	35	35	35	35	35	35	35	0.26	0.00
p1.4.h	45	2.00	45	45	45	45	45	45	45	45	0.66	0.00
p1.4.i	60	2.00	60	60	60	60	60	60	60	60	0.51	0.00
p1.4.j	75	2.60	75	75	75	75	75	75	75	75	0.05	0.00
p1.4.k	100	5.60	100	100	100	100	100	100	100	100	0.14	0.00
p1.4.l	120	5.00	120	120	120	120	120	120	120	120	0.10	0.00
p1.4.m	130	2.40	120	120	120	120	125	125	125	125	0.97	4.00
p1.4.n	155	2.90	145	145	145	145	145	150	150	150	0.88	3.33
p1.4.o	165	3.40	160	160	165	165	165	165	165	165	0.60	0.00
p1.4.p	175	3.60	175	175	175	175	175	175	175	175	0.20	0.00
p1.4.q	190	2.20	180	180	180	180	180	185	185	185	0.42	2.70
p1.4.r	210	1.30	185	185	210	210	210	210	210	210	0.22	0.00
Average:	126.04	6.81	121.56	121.77	122.71	122.81	123.23	123.54	123.65	124.69	0.47	2.04

Table 4. Results for set *p*2.

Instance	BKS ([51])		Our Best Solution (OBS)									Gap (%)
	Reward [a]	Time (s)	Thr_1	Thr_2	Thr_4	Thr_8	Thr_{16}	Thr_{32}	Thr_{64}	Thr_{128} [b]	Time (s)	[a] — [b]
p2.2.a	90	2.30	90	90	90	90	90	90	90	90	0.48	0
p2.2.b	120	1.00	120	120	120	120	120	120	120	120	0.55	0
p2.2.c	140	1.00	130	130	130	130	140	140	140	140	0.41	0
p2.2.d	160	1.00	155	155	160	160	160	160	160	160	0.05	0
p2.2.e	190	3.00	190	190	190	190	190	190	190	190	0.97	0
p2.2.f	200	1.00	200	200	200	200	200	200	200	200	0.03	0
p2.2.g	200	1.00	200	200	200	200	200	200	200	200	0.18	0
p2.2.h	230	1.50	230	230	230	230	230	230	230	230	0.08	0
p2.2.i	230	2.00	230	230	230	230	230	230	230	230	0.25	0
p2.2.j	260	1.00	260	260	260	260	260	260	260	260	0.12	0
p2.2.k	275	6.30	265	265	265	265	275	275	275	275	0.13	0
p2.3.a	70	1.00	70	70	70	70	70	70	70	70	0.99	0
p2.3.b	70	1.00	70	70	70	70	70	70	70	70	0.68	0
p2.3.c	105	1.00	105	105	105	105	105	105	105	105	0.09	0
p2.3.d	105	1.00	105	105	105	105	105	105	105	105	0.38	0
p2.3.e	120	1.00	120	120	120	120	120	120	120	120	0.67	0
p2.3.f	120	1.00	120	120	120	120	120	120	120	120	0.39	0
p2.3.g	145	1.00	140	140	140	140	140	140	140	140	0.44	3.57
p2.3.h	165	1.00	165	165	165	165	165	165	165	165	0.49	0
p2.3.i	200	1.00	200	200	200	200	200	200	200	200	0.96	0
p2.3.j	200	1.00	200	200	200	200	200	200	200	200	0.69	0
p2.3.k	200	1.00	200	200	200	200	200	200	200	200	0.79	0
p2.4.1	10	2.00	10	10	10	10	10	10	10	10	0.72	0

Table 4. *Cont.*

Instance	BKS ([51])		Our Best Solution (OBS)									Gap (%) [a] — [b]
	Reward [a]	Time (s)	Thr_1	Thr_2	Thr_4	Thr_8	Thr_{16}	Thr_{32}	Thr_{64}	Thr_{128} [b]	Time (s)	
p2.4.b	70	1.00	70	70	70	70	70	70	70	70	0.13	0
p2.4.c	70	2.00	70	70	70	70	70	70	70	70	0.94	0
p2.4.d	70	2.00	70	70	70	70	70	70	70	70	0.41	0
p2.4.e	70	2.00	70	70	70	70	70	70	70	70	0.29	0
p2.4.f	105	1.00	105	105	105	105	105	105	105	105	0.62	0
p2.4.g	105	1.00	105	105	105	105	105	105	105	105	0.74	0
p2.4.h	120	1.00	120	120	120	120	120	120	120	120	0.73	0
p2.4.i	120	1.00	120	120	120	120	120	120	120	120	0.66	0
p2.4.j	120	1.00	120	120	120	120	120	120	120	120	0.31	0
p2.4.k	180	1.00	180	180	180	180	180	180	180	180	0.81	0
Average:	140.45	1.43	139.55	139.55	139.70	139.70	140.30	140.30	140.30	140.30	0.49	0.11

7. Analysis of Results

The BKS is obtained in sets $p1$ and $p2$ at least in 75% of the instances (Figure 4). It is obtained even when using a reduced number of threads (between 1 and 8 in many cases). Set $p3$ achieved this outstanding performance in at least 50% of the instances, and sets $p5$, $p6$, and $p7$ in 25% of them. Figure 4 shows gaps for experiments with 128 threads.

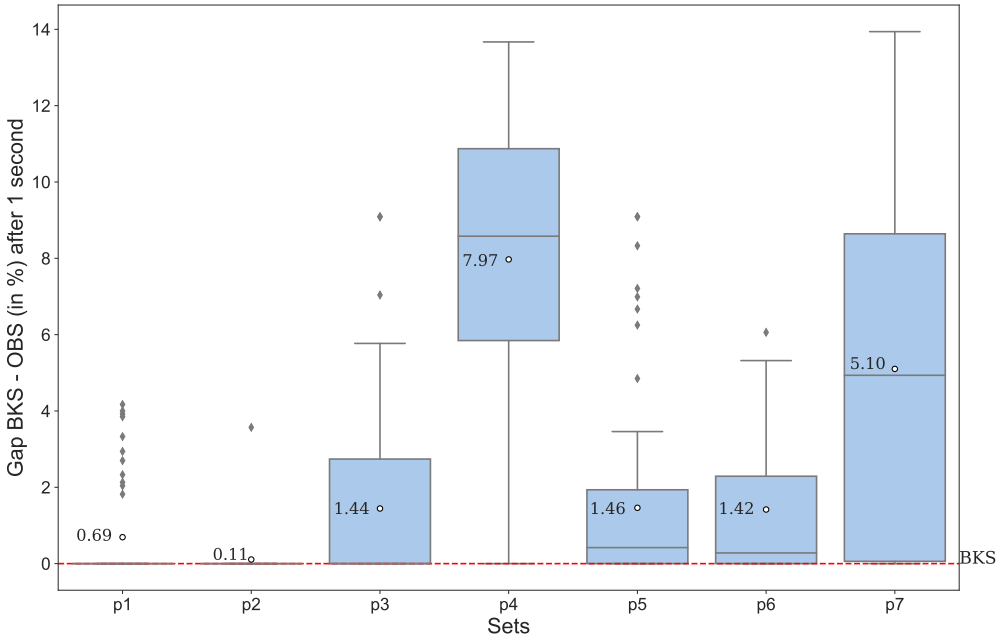

Figure 4. Gaps with respect to the BKS using agile optimization algorithms with 128 threads.

Set $p4$ deserves special attention due to its topology and size (number of nodes, node locations, and rewards). Indeed, it was the most challenging benchmark proposed by Chao et al. [39]. No matter the number of vehicles and the maximum driving range, the resulting gap percentage is higher than the rest of the sets, with a mean value close to 8%. It can also be observed that the larger the size of the instance, the larger the mean gap can be; instances in sets $p4$ and $p7$ have a larger percentage gap (8% and 5%, respectively). Nevertheless, it is also noticeable that the overall percentage gap across the 354 tested instances is just 3%, which is an outstanding result if one considers that it is obtained in less than one second (see Tables 3 and 4). In 54% of the instances, the BKS is achieved by using a reduced number of threads.

Regarding the computational times to reach our best solutions (*OBS*), Figure 5 depicts a boxplot comparing the average computational time—in seconds—invest by our approach in each set, with respect to the computational times used in [51] to reach the BKS. These times are very competitive compared to the average times required to obtain the BKS, approximately 36 s. Notice that by investing an average time of 0.53 s, we obtain an average gap of less than 2.60%, proving that our approach is highly competitive to be used under real-time scenarios. Notice that if we execute our approach serially, we would need, in the worst case, 128 s to reach the same results. Therefore, the benefits of using parallel techniques when dealing with real-time scenarios with a highly dynamic scenario are necessary.

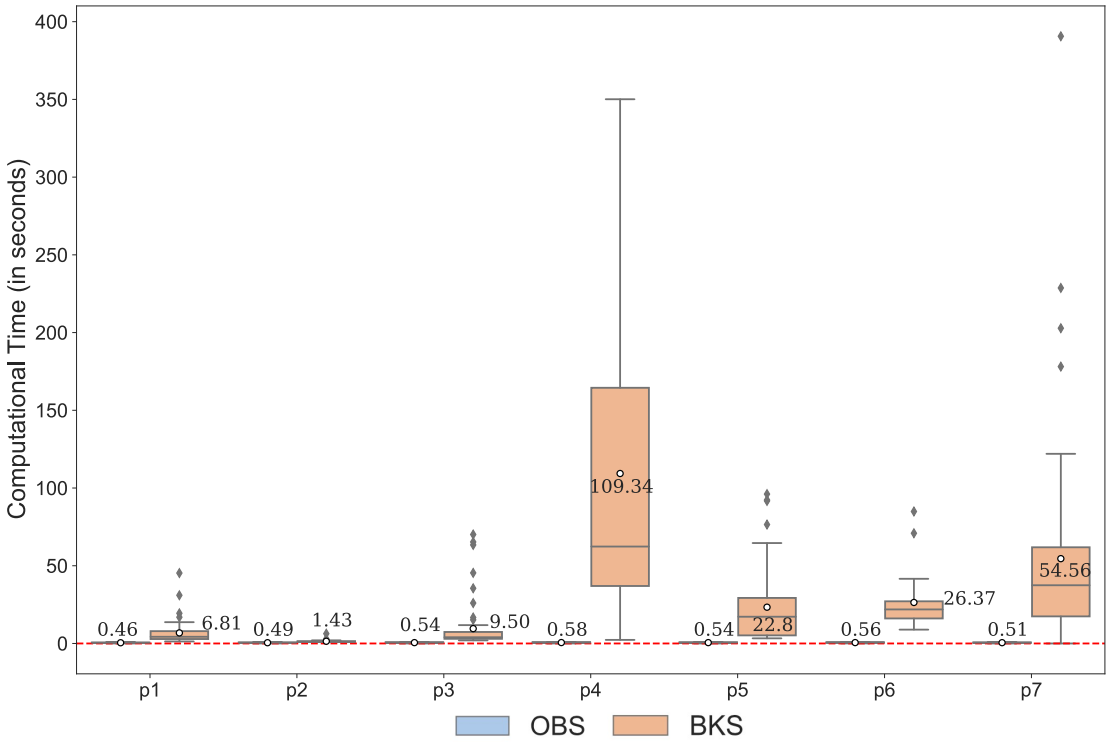

Figure 5. Computational times of our approach with respect to the computational times to reach the BKS.

Finally, Figure 6 illustrates how our agile optimization approach can reduce the gap with respect to the BKS; increasing the number of parallel threads reduces the gap between found solutions and the BKS without increasing the wall-clock time, which is always limited to 1 s.

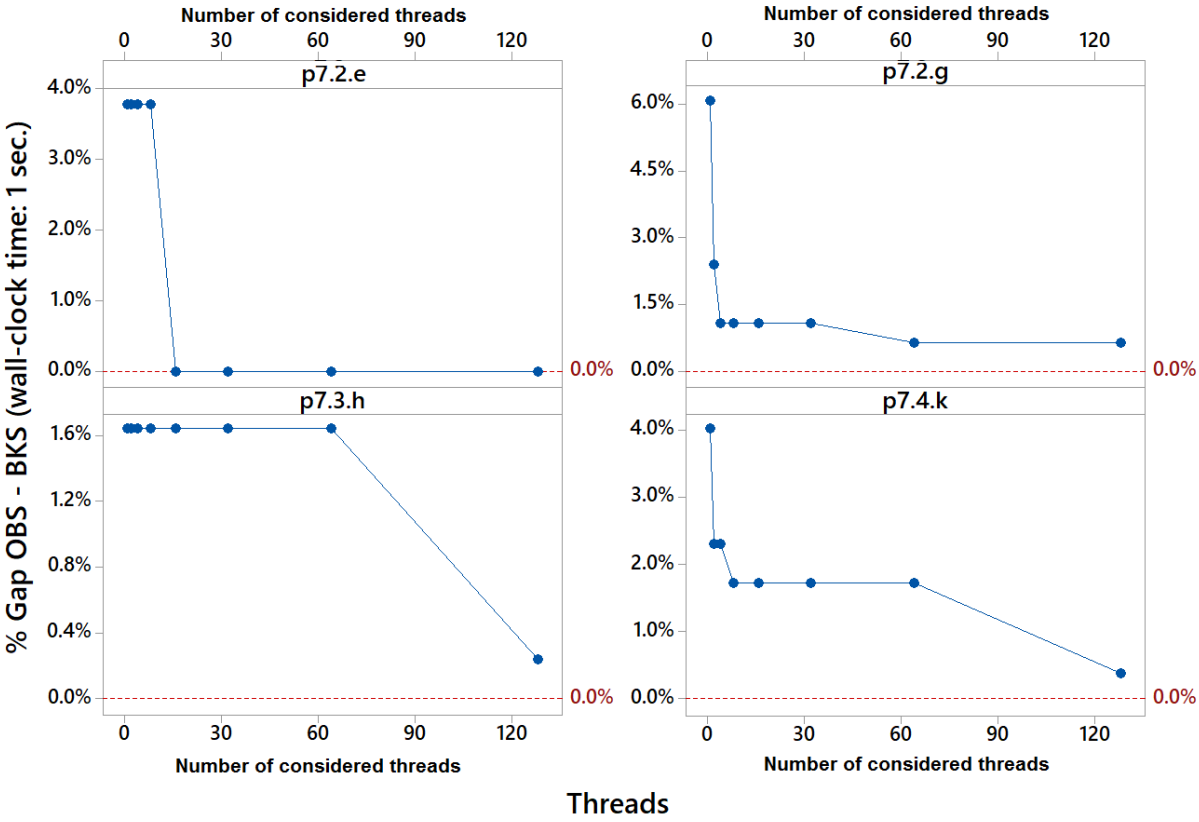

Figure 6. Gap evolution as the number of threads increases for four selected instances.

8. Conclusions

This paper discusses the need for 'agile' optimization in the context of data-driven smart cities, where unmanned aerial vehicles and self-driving vehicles might require solutions to complex problems in real-time (less than a second). The paper uses the team orienteering problem to illustrate these concepts. The team orienteering problem is an NP-hard optimization problem, which challenges the capabilities of exact methods to provide optimal solutions in short computing times. For that reason, the paper proposes a greedy heuristic, which is then extended into a biased-randomized algorithm. This algorithm is then encapsulated into a parallelization framework, allowing running multiple threads in parallel and selecting the best solution. We tested our algorithm on a well-known set of benchmarks for the team orienteering problem. The experiments showed that high-quality solutions could be generated in a few seconds by our algorithm.

Our work could be extended by considering the dynamic environment of the problem. In the dynamic environment, the travel times and rewards change over time. Thus, the constructed routes might be changed during the execution to adapt these changes. The constructed routes and decisions are re-evaluated when a change is detected by sensors, in-route vehicles, video cameras, etc.

Author Contributions: Conceptualization, A.A.J. and J.P.; methodology, A.A.J. and J.P.; software, A.A. and J.P.; validation, C.S., M.N. and M.A.; formal analysis, A.A., C.S. and A.A.J.; investigation, All authors; resources, All authors; data curation, M.A.; writing—original draft preparation, All authors. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work has been partially supported by the Spanish Ministry of Science and Innovation (PID2019-111100RB-C21/AEI/10.13039/501100011033, RED2018-102642-T). We also acknowledge the support of the Erasmus+ Program (2019-I-ES01-KA103-062602).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Griggs, D.; Stafford-Smith, M.; Gaffney, O.; Rockström, J.; Öhman, M.C.; Shyamsundar, P.; Steffen, W.; Glaser, G.; Kanie, N.; Noble, I. Policy: Sustainable Development Goals for People and Planet. *Nature* **2013**, *495*, 305–307. [CrossRef]
2. Nogal, M.; O'Connor, A. Cyber-transportation Resilience. Context and Methodological Framework. In *Resilience and Risk*; Linkov, I., Palma-Oliveira, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 415–426.
3. Le-Dang, Q.; Le-Ngoc, T. Internet of Things (IoT) Infrastructures for Smart cities. In *Handbook of Smart Cities*; Maheswaran, M., Badidi, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2018; pp. 1–30.
4. Wang, X.; Dessouky, M.; Ordonez, F. A Pickup and Delivery Problem for Ridesharing Considering Congestion. *Transp. Lett.* **2016**, *8*, 259–269. [CrossRef]
5. Taş, D. Electric Vehicle Routing with Flexible Time Windows: A Column Generation Solution Approach. *Transp. Lett.* **2021**, *13*, 97–103. [CrossRef]
6. Watts, A.C.; Ambrosia, V.G.; Hinkley, E.A. Unmanned Aircraft Systems in Remote Sensing and Scientific Research: Classification and Considerations of Use. *Remote Sens.* **2012**, *4*, 1671–1692. [CrossRef]
7. Mohamed, N.; Al-Jaroodi, J.; Jawhar, I.; Idries, A.; Mohammed, F. Unmanned Aerial Vehicles Applications in Future Smart Cities. *Technol. Forecast. Soc. Chang.* **2020**, *153*, 119293. [CrossRef]
8. Huang, H.; Savkin, A.V. Optimal Deployment of Charging Stations for Aerial Surveillance by UAVs with the Assistance of Public Transportation Vehicles. *Sensors* **2021**, *21*, 5320. [CrossRef] [PubMed]
9. Zorbas, D.; Pugliese, L.D.P.; Razafindralambo, T.; Guerriero, F. Optimal Drone Placement and Cost-efficient Target Coverage. *J. Netw. Comput. Appl.* **2016**, *75*, 16–31. [CrossRef]
10. Al-Turjman, F.; Zahmatkesh, H.; Al-Oqily, I.; Daboul, R. Optimized Unmanned Aerial Vehicles Deployment for Static and Mobile Targets' Monitoring. *Comput. Commun.* **2020**, *149*, 27–35. [CrossRef]
11. Hassanalian, M.; Mirzaeinia, A.; Lee, K. Smart Cities and Organizing the Drones' Applications in Urban Areas: NE ST (Networking, Efficient, Strategies). In Proceedings of the 2020 AIAA Scitech Forum, Orlando, FL, USA, 6–10 January 2020; p. 1944.
12. Torabbeigi, M.; Lim, G.J.; Ahmadian, N.; Kim, S.J. An Optimization Approach to Minimize the Expected Loss of Demand Considering Drone Failures in Drone Delivery Scheduling. *J. Intell. Robot. Syst.* **2021**, *102*, 22. [CrossRef]
13. Chowdhury, S.; Shahvari, O.; Marufuzzaman, M.; Li, X.; Bian, L. Drone Routing and Optimization for Post-disaster Inspection. *Comput. Ind. Eng.* **2021**, *159*, 107495. [CrossRef]
14. Xia, J.; Wang, K.; Wang, S. Drone Scheduling to Monitor Vessels in Emission Control Areas. *Transp. Res. Part Methodol.* **2019**, *119*, 174–196. [CrossRef]
15. Mirjalili, S. Special Issue on "Real-world Optimization Problems and Meta-heuristics". *Neural Comput. Appl.* **2020**, *32*, 11965–11966. [CrossRef]
16. Azadeh, A.; Farrokhi-Asl, H. The Close–Open Mixed Multi Depot Vehicle Routing Problem Considering Internal and External Fleet of Vehicles. *Transp. Lett.* **2019**, *11*, 78–92. [CrossRef]
17. Juan, A.A.; Keenan, P.; Martí, R.; McGarraghy, S.; Panadero, J.; Carroll, P.; Oliva, D. A Review of the Role of Heuristics in Stochastic Optimisation: From Metaheuristics to Learnheuristics. *Ann. Oper. Res.* **2021**. [CrossRef]
18. Ferone, D.; Gruler, A.; Festa, P.; Juan, A.A. Enhancing and extending the classical GRASP framework with biased randomisation and simulation. *J. Oper. Res. Soc.* **2019**, *70*, 1362–1375. [CrossRef]
19. Golub, G.H.; Ortega, J.M. *Scientific Computing: An Introduction with Parallel Computing*; Elsevier: Amsterdam, The Netherlands, 2014.
20. Belloso, J.; Juan, A.A.; Faulin, J. An iterative biased-randomized heuristic for the fleet size and mix vehicle-routing problem with backhauls. *Int. Trans. Oper. Res.* **2019**, *26*, 289–301. [CrossRef]
21. Ferrer, A.; Guimarans, D.; Ramalhinho, H.; Juan, A.A. A BRILS metaheuristic for non-smooth flow-shop problems with failure-risk costs. *Expert Syst. Appl.* **2016**, *44*, 177–186. [CrossRef]
22. Dominguez, O.; Juan, A.A.; de La Nuez, I.; Ouelhadj, D. An ILS-biased randomization algorithm for the two-dimensional loading HFVRP with sequential loading and items rotation. *J. Oper. Res. Soc.* **2016**, *67*, 37–53. [CrossRef]
23. Ferone, D.; Hatami, S.; González-Neira, E.M.; Juan, A.A.; Festa, P. A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem. *Int. Trans. Oper. Res.* **2020**, *27*, 1368–1391. [CrossRef]
24. Roberge, V.; Tarbouchi, M. Parallel Algorithm on GPU for Wireless Sensor Data Acquisition Using a Team of Unmanned Aerial Vehicles. *Sensors* **2021**, *21*, 6851. [CrossRef]
25. Yelmewad, P.; Talawar, B. Parallel Version of Local Search Heuristic Algorithm to Solve Capacitated Vehicle Routing Problem. *Clust. Comput.* **2021**, *24*, 3671–3692. [CrossRef]

26. Abbasi, M.; Rafiee, M.; Khosravi, M.R.; Jolfaei, A.; Menon, V.G.; Koushyar, J.M. An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems. *J. Cloud Comput.* **2020**, *9*, 6. [CrossRef]
27. Fava, L.P.; Furtado, J.C.; Helfer, G.A.; Barbosa, J.L.V.; Beko, M.; Correia, S.D.; Leithardt, V.R.Q. A Multi-Start Algorithm for Solving the Capacitated Vehicle Routing Problem with Two-Dimensional Loading Constraints. *Symmetry* **2021**, *13*, 1697. [CrossRef]
28. Huang, C.; Huang, G.; Liu, W.; Wang, R.; Xie, M. A parallel joint optimized relay selection protocol for wake-up radio enabled WSNs. *Phys. Commun.* **2021**, *47*, 101320. [CrossRef]
29. Panadero, J.; de Armas, J.; Currie, C.S.; Juan, A.A. A simheuristic approach for the stochastic team orienteering problem. In Proceedings of the 2017 Winter Simulation Conference (WSC), Las Vegas, NV, USA, 3–6 December 2017; pp. 3208–3217.
30. Panadero, J.; Juan, A.A.; Bayliss, C.; Currie, C. Maximising reward from a team of surveillance drones: A simheuristic approach to the stochastic team orienteering problem. *Eur. J. Ind. Eng.* **2020**, *14*, 485–516. [CrossRef]
31. Reyes-Rubiano, L.; Juan, A.A.; Bayliss, C.; Panadero, J.; Faulin, J.; Copado, P. A biased-randomized learnheuristic for solving the team orienteering problem with dynamic rewards. *Transp. Res. Procedia* **2020**, *47*, 680–687. [CrossRef]
32. Mitchell, J.S. Geometric Shortest Paths and Network Optimization. In *Handbook of Computational Geometry*; Elsevier: Amsterdam, The Netherlands, 2000; pp. 633–701.
33. Golden, B.L.; Levy, L.; Vohra, R. The Orienteering Problem. *Nav. Res. Logist. (NRL)* **1987**, *34*, 307–318. [CrossRef]
34. Laporte, G.; Martello, S. The Selective Travelling Salesman Problem. *Discret. Appl. Math.* **1990**, *26*, 193–207. [CrossRef]
35. Feillet, D.; Dejax, P.; Gendreau, M. Traveling Salesman Problems with Profits. *Transp. Sci.* **2005**, *39*, 188–205. [CrossRef]
36. Gunawan, A.; Lau, H.C.; Vansteenwegen, P. Orienteering Problem: A Survey of Recent Variants, Solution Approaches and Applications. *Eur. J. Oper. Res.* **2016**, *255*, 315–332. [CrossRef]
37. Vansteenwegen, P.; Gunawan, A. *Orienteering Problems: Models and Algorithms for Vehicle Routing Problems with Profits*; Springer: Cham, Switzerland, 2019.
38. Archetti, C.; Hertz, A.; Speranza, M. Metaheuristics for the Team Orienteering Problem. *J. Heurist.* **2007**, *13*, 49–76. [CrossRef]
39. Chao, I.M.; Golden, B.L.; Wasil, E.A. A Fast and Effective Heuristic for the Orienteering Problem. *Eur. J. Oper. Res.* **1996**, *88*, 475–489. [CrossRef]
40. Boussier, S.; Feillet, D.; Gendreau, M. An Exact Algorithm for the Team Orienteering Problem. *4OR* **2007**, *5*, 211–230. [CrossRef]
41. Keshtkaran, M.; Ziarati, K.; Bettinelli, A.; Vigo, D. Enhanced Exact Solution Methods for the Team Orienteering Problem. *Int. J. Prod. Res.* **2016**, *54*, 591–601. [CrossRef]
42. Bianchessi, N.; Mansini, R.; Speranza, M.G. A Branch-and-Cut Algorithm for the Team Orienteering Problem. *Int. Trans. Oper. Res.* **2018**, *25*, 627–635. [CrossRef]
43. Butt, S.E.; Ryan, D.M. An Optimal Solution Procedure for the Multiple Tour Maximum Collection Problem Using Column Generation. *Comput. Oper. Res.* **1999**, *26*, 427–441. [CrossRef]
44. Tang, H.; Miller-Hooks, E. A Tabu Search Heuristic for the Team Orienteering Problem. *Comput. Oper. Res.* **2005**, *32*, 1379–1407. [CrossRef]
45. Vansteenwegen, P.; Souffriau, W.; Berghe, G.; Oudheusden, D. A Guided Local Search Metaheuristic for the Team Orienteering Problem. *Eur. J. Oper. Res.* **2009**, *196*, 118–127. [CrossRef]
46. Ke, L.; Archetti, C.; Feng, Z. Ants can solve the team orienteering problem. *Comput. Ind. Eng.* **2008**, *54*, 648–665. [CrossRef]
47. Bouly, H.; Dang, D.; Moukrim, A. A Memetic Algorithm for the Team Orienteering Problem. *4OR-Q. J. Oper. Res.* **2010**, *8*, 49–70. [CrossRef]
48. Lin, S.W. Solving the Team Orienteering Problem Using Effective Multi-start Simulated Annealing. *Appl. Soft Comput. J.* **2013**, *13*, 1064–1073. [CrossRef]
49. Dang, D.C.; Guibadj, R.N.; Moukrim, A. An Effective PSO-inspired Algorithm for the Team Orienteering Problem. *Eur. J. Oper. Res.* **2013**, *229*, 332–344. [CrossRef]
50. Ferreira, J.; Quintas, A.; Oliveira, J.; Pereira, G.; Dias, L. Solving the Team Orienteering Problem: Developing a Solution Tool Using a Genetic Algorithm Approach. In *Soft Computing in Industrial Applications. Advances in Intelligent Systems and Computing*; Springer: Berlin/Heidelberg, Germany, 2014; Volume 223, pp. 365–375.
51. Ke, L.; Zhai, L.; Li, J.; Chan, F.T. Pareto Mimic Algorithm: An Approach to the Team Orienteering Problem. *Omega* **2016**, *61*, 155–166. [CrossRef]
52. Tsakirakis, E.; Marinaki, M.; Marinakis, Y.; Matsatsinis, N. A Similarity Hybrid Harmony Search Algorithm for the Team Orienteering Problem. *Appl. Soft Comput.* **2019**, *80*, 776–796. [CrossRef]
53. Gruler, A.; Panadero, J.; de Armas, J.; Moreno, J.A.M.; Juan, A.A. Combining variable neighborhood search with simulation for the inventory routing problem with stochastic demands and stock-outs. *Comput. Ind. Eng.* **2018**, *123*, 278–288. [CrossRef]
54. Pagès-Bernaus, A.; Ramalhinho, H.; Juan, A.A.; Calvet, L. Designing e-commerce supply chains: A stochastic facility–location approach. *Int. Trans. Oper. Res.* **2019**, *26*, 507–528. [CrossRef]
55. Guimarans, D.; Dominguez, O.; Panadero, J.; Juan, A.A. A simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times. *Simul. Model. Pract. Theory* **2018**, *89*, 1–14. [CrossRef]
56. Clarke, G.; Wright, J.W. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Oper. Res.* **1964**, *12*, 568–581. [CrossRef]
57. Estrada-Moreno, A.; Savelsbergh, M.; Juan, A.A.; Panadero, J. Biased-randomized iterated local search for a multiperiod vehicle routing problem with price discounts for delivery flexibility. *Int. Trans. Oper. Res.* **2019**, *26*, 1293–1314. [CrossRef]

58. Martí, R.; Resende, M.G.; Ribeiro, C.C. Multi-start Methods for Combinatorial Optimization. *Eur. J. Oper. Res.* **2013**, *226*, 1–8. [CrossRef]
59. Régin, J.C.; Rezgui, M.; Malapert, A. Embarrassingly Parallel Search. In *Principles and Practice of Constraint Programming*; Schulte, C., Ed.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 596–610.
60. Parhami, B. *Introduction to Parallel Processing: Algorithms and Architectures*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006.
61. Gulati, K.; Khatri, S.P. GPU Architecture and the CUDA Programming Model. In *Hardware Acceleration of EDA Algorithms*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 23–30.
62. Solihin, Y. *Fundamentals of Parallel Multicore Architecture*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2015.