1. Introduction

2. Related literature

3. Problem context and assumptions

4. The mathematical modeling

5. Solution strategy

6. Implementation and results

6.1. Results obtained

In the tests performed, we used a 64-bit, 16GB, 3.6GHz, eight-core processor with *Linux Ubuntu 22.04.1 LTS 64-bit* as the operational system and *Python 3.10.4* as the programming language. We also used the well-known solver *Gurobi* (www.gurobi.com), version 9.5.2.

We ran Algorithm **??** in the 5 scenarios described in Table **??**, considering 6 methods for node-by-node solution: Gurobi (see **??**), ACO, NMO, TS, GRASP, and *Shims* (Algorithm **??**). In generating the items in each node, we consider 3 values for the parameter *surplus*. The results obtained for the function f, with the corresponding run time in seconds, are shown in Tables 1 ($surplus = 1.2$), 2 ($surplus = 1.5$) and 3 ($surplus = 2.0$). For each *method*, *scenario* and *surplus*, 7 different instances were generated. Therefore, 105 tests (5 scenarios $\times$ 3 volumes $\times$ 7 instances) were performed for each method.

The average values were presented for f_π and, for the run time, the worst result obtained. To facilitate the comparison between the methods, we added a last column in these tables, where two values are indicated:

- **Normalized**: value between 0 and 1, which corresponds to the ratio between the sum of f values obtained by the method in all scenarios and the sum of the best values obtained among all methods in all scenarios. The higher the value of **Normalized**, the closer the method approached the best solutions found.
- **Speed-up**: ratio of the sums of the worst run times of all scenarios and the sum of the method run times in all scenarios. The method with the highest **Speed-up** is the fastest.

In each *scenario*, we indicate in bold the best value of f found. In each table, we also indicate in bold the best **Normalized** and **Speed-up** values.

run time (s) is the average time to solve a problem instance.

As the MIP solver was not capable of solving all scenarios in polynomial time or reached the computer's Read-Access Memory (RAM, 16 GiB) too early, we decided to set the Gurobi *MIPgap* parameter to 1% to control the minimal quality of the returned solutions and abbreviate the solution time. From now on, we will refer to this Gurobi-specific model as *Gurobi** . For more details on the Gurobi *MIPgap*, refer to [1].

This decision hinders the MIP solver from reaching optimal solutions in favor of the possibility of result comparisons with the *Shims* heuristic.

Table 1: Solutions with volume *surplus* = 1.2 and 3600s as time limit.

Tours	*method*	**Scenarios**	**1**	**2**	**3**	**4**	**5**	**Normalized** **Speed-up**
π_{TSP_1} π_{TSP_2}	*Gurobi**	f	8.52	11.70	13.07	13.54	12.44	1.00
		run time (s)	17	17	16	17	26	1.0
	Shims	f	8.54	11.63	12.97	13.43	12.36	0.99
		run time (s)	1	1	2	2	2	11.6
S_K	*Gurobi**	f	8.52	12.25	13.32	14.61	x	1.00
		run time (s)	25	36	121	297	x	1.0
	Shims	f	8.54	12.26	13.23	14.53	13.42	0.99
		run time (s)	1	2	6	17	106	18.4

**MIPgap* parameter set to 1%. The f values are $\pm 1\%$ distant from the optimal solution

97% of 16GiB was the RAM state before we aborted the *Gurobi** execution on solving scenario 5 in Table 1.

Table 2: Solutions with volume *surplus* = 1.5 and 3600s as time limit.

Tours	*method*	**Scenarios**	**1**	**2**	**3**	**4**	**5**	**Normalized Speed-up**
π_{TSP_1} π_{TSP_2}	*Gurobi**	f	11.80	16.74	18.04	18.86	16.92	1.00
		run time (s)	38	30	27	29	81	1.00
	Shims	f	11.83	16.74	18.01	18.89	16.91	1.00
		run time (s)	1	2	2	3	3	20.5
S_K	*Gurobi**	f	11.81	17.01	18.25	20.59	18.36	1.00
		run time (s)	53	61	193	470	2241	1.0
	Shims	f	11.83	17.00	18.21	20.45	18.00	0.99
		run time (s)	1	2	9	26	157	15.5

Table 3: Solutions with volume *surplus* = 2.0 and 3600s as time limit.

Tours	*method*	**Scenarios**	**1**	**2**	**3**	**4**	**5**	**Normalized Speed-up**
π_{TSP_1} π_{TSP_2}	*Gurobi**	f	17.75	24.39	26.46	27.09	24.00	1.00
		run time (s)	162	95	74	68	67	1.0
	Shims	f	17.78	24.40	26.36	27.01	23.94	0.99
		run time (s)	2	3	3	4	5	27.4
S_K	*Gurobi**	f	17.75	25.24	26.46	29.11	x	1.00
		run time (s)	151	123	319	689	x	1.0
	Shims	f	17.78	25.25	26.38	28.98	26.08	1.00
		run time (s)	2	3	13	37	229	23.3

*Gurobi** could not solve scenario 5 in Table 3 because $3600/K! = 0.71s$ is not enough time for *Gurobi** to find a feasible node solution.

6.1.1. Time limits comparisons

Table 4: Solutions with different time limitations for S_K tours.

	Volume *surplus*		1.2		1.5		2.0	
method	**Time limit (s)**	**Scenarios**	**4**	**5**	**4**	**5**	**4**	**5**
*Gurobi**	1200	f	14.67	x	20.64	x	27.97	x
		run time (s)	217	x	304	x	579	x
	2400	f	14.59		20.60	x	29.10	x
		run time (s)	313		327	x	543	x
	3600	f	14.61	x	20.59		29.11	x
		run time (s)	297	x	470		689	x
Shims	1200	f	14.53	13.42	20.45	18.00	28.98	26.08
		run time (s)	18	109	26	160	37	203
	2400	f	14.53	13.42	20.45	18.00	28.98	26.08
		run time (s)	18	107	25	159	37	192
	3600	f	14.53	13.42	20.45	18.00	28.98	26.08
		run time (s)	17	106	26	157	37	181
		Normalized Speed-up	0.99	x	0.99	x	1.00	x
			15.6	x	14.3	x	16.3	x

6.1.2. Proposed K! strategy assessment

Another important test is to compare the best tour score/cost ratio with the least expensive tour score/cost ratio. This is important to see how far our solution strategy is from the simplest form, just solving a TSP.

For this achievement, we ran *Shims* in 5 scenarios with 7 instances each, and 3 volume surpluses, collecting the averages of the best tour result and the least cost tour result.

Table 5: *Shims* best and least cost tour results

surplus	**Tour**	**1**	**2**	**3**	**4**	**5**	**Sum**	**Improvement**
1.2	Least cost	8.54	11.63	12.97	13.43	12.36	58.93	
	Best in $K!$	8.54	12.26	13.23	14.53	13.42	61.98	5.2%
1.5	Least cost	10.89	11.83	16.74	18.01	18.89	76.36	
	Best in $K!$	11.33	11.83	17.00	18.21	20.45	78.82	3.2%
2.0	Least cost	17.78	24.40	26.36	27.01	23.94	119.49	
	Best in $K!$	17.78	25.25	26.38	28.98	26.08	124.47	4.2%
						K! strategy advantage		**4.2%**

As the selection of the best tour, considering what was collected and delivered in each node, and considering the cost increase due to CG deviation, was better than the option for a simple TSP solution (*Least cost* results), this confirms that our solution strategy was adequate.

It is important to report that the best tour found in each test was seldom the tour with the lowest cost (the minimal cost tour without considering the CG deviation), but any of the first 25% of the best tours. This indicates that our strategy of enumerating and solving all tours was assertive, although it is possible to devise a more proactive strategy to discard most of the higher-cost tours and improve performance. This must be one of the next steps in this research.

If a posterior application of this method requires 8, 9, or more nodes, it is possible to use this 25% as a threshold to discard the higher-cost tours and keep the run time within an operational acceptable limit for the client.

6.1.3. Final considerations on the results

Most of the *Gurobi** executions consumed over 12 GiB in total, and considering that the computer used in the experiments has a standard RAM consumption when idle of 3.5 GiB (with the Python Interface Development Environment (IDE), a PDF viewer, a system monitor, and a *LaTeX* IDE simultaneously open), the actual RAM consumption of *Gurobi** was over 8.5 GiB.

All *Shims* executions consumed at most 1.5 GiB, requiring 5 GiB of total memory. These details are important for a future application to adequately dimension the computer to be used.

References

[1] Matthias Miltenberger. *What is the MIPGap?* `https://support.gurobi.com/hc/en-us/articles/8265539575953-What-is-the-MIPGap-`, 2023.