Local Search Heuristic for the Optimisation of Flight Connections

Maab Alrasheed*, Wafaa Mohammed*, Yaroslav Pylyavskyy†, Ahmed Kheiri†
*University of Khartoum, Faculty of Engineering
Department of Electrical and Electronic Engineering
Algamaa Street, P.O. Box 321, Khartoum, Sudan
{maabnimir, wafamoh97}@gmail.com
†Lancaster University, Department of Management Science
Lancaster University Management School
Lancaster, LA1 4YX, United Kingdom
{y.pylyavskyy, a.kheiri}@lancaster.ac.uk

***Abstract*—Kiwi.com proposed a real-world NP-hard optimisation problem with a focus on air travelling services, determining the cheapest connection between specific areas. Despite some similarities with the classical TSP problem, more complexity is involved that makes the problem unique. It is *Time-dependent*, *Asymmetric* and involves areas that contain sets of cities from which exactly one is visited. In addition to this, infeasibility adds more complexity to the problem since there are no flights available between specific points in the network for certain days. While solving such computationally difficult problems, exact methods often fail, particularly when the problem instance size increases; Then alternative approaches, such as heuristics, are preferred in problem solving. In this study, we present an effective local search method for solving Kiwi.com problem. The empirical results show the success of the approach, which embeds four simple operators, on most of the released instances.**

***Index Terms*—Local Search, Optimisation, Metaheuristics, Computational Design, Travelling Salesman Problem**

I. Introduction

Travelling salesman problem (TSP) is a well-known problem in the field of computer science and operational research. Many variations of TSP have been created in order to fit and tackle real-world problems. Kiwi.com problem is a modified version of the ordinary TSP and it is described as follows: Given a set of n areas, the airports in each area, the starting airport and the costs of travelling between those airports, the goal is to find the cheapest possible route that visits each area exactly once.

There are several variants of TSP formulations [1]:

- General: The cost is arbitrarily assigned between cities.
- Metric: The cost c is metric and satisfies the triangle inequality; $\forall i, j, k, c_{ik} \leq c_{ij} + c_{jk}$.
- Symmetric TSP: The cost of travelling from city i to city j is the same as travelling from city j to city i; i.e. $c_{ij} = c_{ji}$.
- Asymmetric TSP: The cost of travelling from city i to city j is not the same as travelling from city j to city i; $c_{ij} \neq c_{ji}$.
- TSP with multiple visits (TSPM): It is allowed to visit cities more than once.
- Open tour TSP: The salesman does not have to end the tour where it started.
- Multiple TSP: Instead of only one salesman, multiple salesmen are allowed [2].
- Time-dependent TSP: The travel cost depends on the distance and the day of travel [3].
- Maximum benefit TSP (MBTSP): Salesman derives some benefit from visiting the cities [4].
- Prize collecting TSP (PCTSP): Salesman gets a prize for every visited city and pays a penalty for every city not visited [5].

Numerous of algorithms have been developed for solving TSP. Such algorithms are classified into:

1) Exact methods, such as Branch-and-Cut. However, exact methods often fail particularly when the problem instance size increases [6].
2) Heuristic algorithms which are used to provide near optimal solutions for TSP within reasonable time. Examples include local search algorithms [7], genetic algorithms [8], simulated annealing [6], hyper-heuristics [9], [10], artificial neural network [11], ant colony optimisation [12] and particle swarm optimisation [13].

Thanaboon and Pisut [7] have discussed the time-dependent asymmetric TSP with time window (a country may have to be visited within a given time period) and precedence constraints (a country may have to be visited immediately after a predefined preceding country) in Air Travel. They suggested three algorithms to solve the problem: MNN (modified nearest neighbour), LS-SWAP (local search algorithm with swap operator) and LS-INSERT (local search algorithm with insert operator). They used small instances with around 20 cities to test their algorithms. Their results indicated that the local search algorithms perform better than the MNN algorithm in terms of the total cost and execution time. LS-INSERT performs slightly better on 'easy' instances while LS-SWAP is more recommended for 'hard' instances.

The paper is structured as follows: Section II describes the Kiwi.com problem. In Section III we discuss the local

search algorithm and the four operators embedded. Section IV presents the results. Finally, the conclusion and suggested further research are provided in Section V.

II. Problem description

Kiwi.com problem requires the minimisation of the travelling cost by finding the best possible flight route for a given number of areas, where an area is a set of cities (airports). Costs differ according to the direction and day of travel. The trip begins from a given city and everyday exactly one city of each area is visited. Throughout the trip, the arrival city is also the departure city for each area. This means that it is not allowed to arrive to a city and continue the trip by departing from another city of the same area. The trip ends in the area (not necessarily the city) where it began.

A simple problem instance consisting of 4 areas and 5 airports is presented in Figure 1.

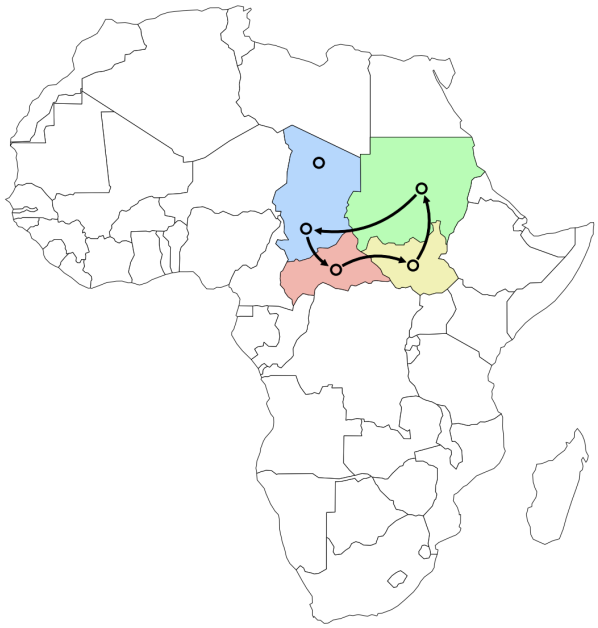

Fig. 1. Simple problem instance with a possible solution

Mathematically, let $Area = \{area_1, area_2, \ldots, area_n\}$ be a set of n areas, where each area $r \in Area$ is composed of a set of airports $\{airport_1, airport_2, \ldots\}$; and let c_{ij}^d be the flight cost between the departure airport i and the arrival airport j on day d, which has two properties: c_{ij}^d is not necessarily equal c_{ji}^d (i.e. the problem is asymmetric); and for $d1 \neq d2$, c_{ij}^{d1} is not necessarily equal c_{ij}^{d2}, (i.e. the problem is time dependent). The objective of the problem is to find the best possible flight route that connects all the given areas and minimises the cost within the time given, subject to the following constraints:

- The trip starts from the starting airport (city) given.
- Exactly one city is visited in each area (but we can choose which one).
- Every day a different area is visited.
- The trip continues from the arrival airport.
- The entire trip ends in any airport of the area where it began.

III. Local search algorithm

Local search algorithm (see Algorithm 1) moves from one solution to another in the search space by applying an operator (e.g. swap operator) until a better solution is found [14].

Algorithm 1: Local search algorithm

```
Let O represent the set of operators
Let S represent the current solution
Let S_new represent the new solution
S ← Initialise();
repeat
    O_i ← Select(O);
    S_new ← ApplyOperator(O_i, S);
    if S_new isNotWorseThan S then
        S ← S_new;
    end
until TimeLimit;
```

A solution is represented by a pair $S = (\pi, airport)$, where π is a permutation of the areas and $airport$ is a set of airports. Two methods were implemented to generate initial solutions, including *random* method and *greedy*. The random method lists the areas and airports in the order they were provided in the input instance. The greedy method chooses the next airport that generates the lowest cost from the remaining airports to visit. Both methods do not guarantee the feasibility of the solution. Hence, it is repaired using the proposed local search algorithm. Following the initialisation stage, the same local search method uses the remaining time left to improve the quality (cost) of the solution.

The local search method manages the following four operators: (i) *swap* areas then change their airports with a probability of 50%; (ii) *insert* an area then change their airports with a probability of 50%; (iii) *reverse* then change their airports with a probability of 50%; and (iv) *change airport*.

The time limit that the algorithm was allowed to run depends on the size of the problem instance, as suggested by Kiwi.com:

- For small instances, i.e. instances where the number of areas is less than or equal 20 and the total number of airports is less than 50 - the time limit is 3s.
- For medium instances, i.e. instances where the number of areas is less than or equal 100 and the total number of airports is less than 200 - the time limit is 5s.
- Otherwise the time limit is 15s.

IV. Results

Our experiments were performed on an i7-4710HQ Intel Processor at 2.50GHz with 16.00GB RAM and the algorithms have been implemented using VC++ 2017. The summary of

TABLE I
THE PERFORMANCE OF THE LOCAL SEARCH ALGORITHMS OVER 10 RUNS. WE USE "-" TO DENOTE THE CASE WHEN NO FEASIBLE SOLUTION IS FOUND WITHIN THE TIME LIMIT

	Characteristics of the Dataset				Random			Greedy		
Name	areas	airports	airports in areas	time	best	average	std.	best	average	std.
Instance-1	10	10	1 (min) – 1 (max)	3	1396	1434.3	41.0	1396	1447.9	33.3
Instance-2	10	15	1 (min) – 2 (max)	3	1498	1498	0.0	1498	1498	0.0
Instance-3	13	38	1 (min) – 6 (max)	3	7672	7858.1	197.3	7951	8035	55.0
Instance-4	40	99	1 (min) – 5 (max)	5	14045	14552.7	389.3	14066	14425.6	237.2
Instance-5	46	138	3 (min) – 3 (max)	5	837	930.9	58.6	739	890.5	129.5
Instance-6	96	192	2 (min) – 2 (max)	5	3021	3965.1	436.6	3791	4325.6	382.1
Instance-7	150	300	1 (min) – 6 (max)	15	32705	38654.3	6991.1	32534	34698.5	3724.2
Instance-8	200	300	1 (min) – 4 (max)	15	7712	9747.3	1787.1	4041	4068.1	18.5
Instance-9	250	250	1 (min) – 1 (max)	15	221188	233509	10129.4	82242	96695.4	9864.8
Instance-10	300	300	1 (min) – 1 (max)	15	122719	129511	8395.9	87462	116301	11291.5
Instance-11	150	200	1 (min) – 4 (max)	15	70579	73743	2393.1	49453	66399.2	7766.4
Instance-12	200	250	1 (min) – 4 (max)	15	113084	118149	3577.6	70082	99283.2	17323.5
Instance-13	250	275	1 (min) – 3 (max)	15	-	-	-	-	-	-
Instance-14	300	300	1 (min) – 1 (max)	15	-	-	-	-	-	-

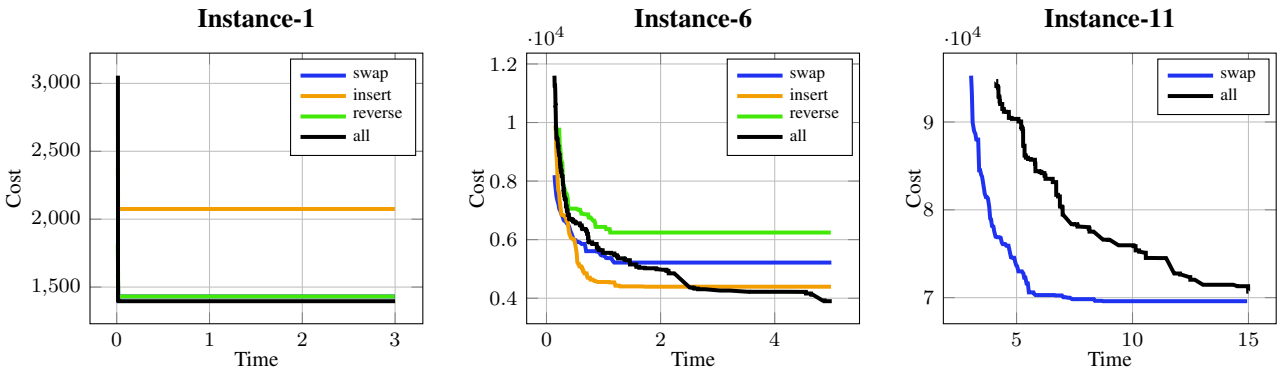

Fig. 2. Plots of the cost versus time while solving Instance-1, Instance-6 and Instance-11, respectively

experimental results are presented in Table I including the characteristics of the Kiwi.com dataset. The experiments were repeated 10 times for each problem instance using different random seed values. The table illustrates the best and average costs found along with the standard deviation for both initialisation methods (i.e. random method and greedy approach) over 10 runs. The experiments suggest that initialising the solution using random method is better for small instances, while greedy approach often leads to better solutions for medium and large size instances.

A. An analysis of the local search method

Figure 2 shows the difference in cost reduction between the application of the four operators separately and their jointly application. The results are presented for Instance-1 (small instance), Instance-6 (medium instance) and Instance-11 (large instance). Figure 3 shows the utilisation rate in cost reduction of each operator.

As far as the four operators are concerned, results suggest that:

- Swap operator performs a change in only two positions in the solution. When applied separately, it performs very well in reducing the cost for all types of instances (small,

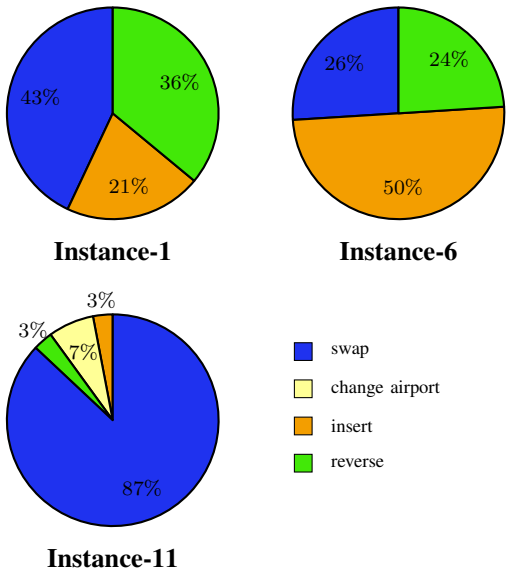

Fig. 3. Utilisation rate of each operator

medium and large) and is particularly effective for large instances. When combined with the other operators, its utilisation rate in cost reduction is high.

- The performance of the change airport operator depends on the number of airports in the areas. However, all instances in our experiments have a limited number of airports in each area. Thus, the contribution in cost reduction of this operator is relatively small. When applied separately, this operator has failed to obtain feasible solutions.
- The performance of the insert and reverse operators highly depends on the two random positions selected. When the distance between the two selected positions is large they perform a big change in the solution, and thus increasing the probability of obtaining an infeasible solution. Hence, they contribute more in small and medium instances. In addition, insert operator seems to outperform the other operators for medium size instances in terms of utilisation rate.

Further analysis of the results confirm the following:

- For small instances, swap and reverse operators perform very well and provide good solutions. Insert operator has the worst performance among the operators. The best solution, however, is achieved when all operators are combined together.
- For medium instances, insert operator is the best compared to the other operators. This is because the number of areas in these instances are neither too small nor too large. Hence, the change in the solution is also moderate and provides good results with relatively low chance of infeasibility. Yet again, the combination of operators provide the best solution.
- For large instances, the swap operator is the best. In contrast to small and medium instances swap operator outperforms the combination of all operators. However, an essential drawback is that the whole algorithm performance is poor for large instances.

V. Conclusion

In this study, we proposed a local search algorithm with four operators (swap, insert, change airport and reverse) to solve Kiwi.com problem. Our research has shown that the algorithm provides essentially improved solutions for all types of instances. Overall, the combination of all operators is suggested for small and medium instances. The main contributors are swap and reverse operators and insert operator for small and medium instances respectively. For large instances the swap operator is recommended. However, for such instances the algorithm lacks time efficiency. Furthermore, based on our findings the use of greedy approach for initialising the solutions of medium and large size instances leads to improved performance. The application of advanced algorithms, such as hyper-heuristics [15], [16] and evolutionary algorithms [17]–[19], is suggested for further research in order to escape local minima and achieve improved solutions for medium and large instances.

References

[1] R. Rasmussen, "TSP in spreadsheets - a guided tour," *International Review of Economics Education*, vol. 10, no. 1, pp. 94–116, 2011.

[2] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[3] K. R. Fox, B. Gavish, and S. C. Graves, "Technical note—an n-constraint formulation of the (time-dependent) traveling salesman problem," *Operations Research*, vol. 28, no. 4, pp. 1018–1021, 1980.

[4] C. Malandraki and M. S. Daskin, "The maximum benefit chinese postman problem and the maximum benefit traveling salesman problem," *European Journal of Operational Research*, vol. 65, no. 2, pp. 218–234, 1993.

[5] E. Balas, "The prize collecting traveling salesman problem," *Networks*, vol. 19, no. 6, pp. 621–636, 1989.

[6] M. M. Abid and I. A. Muhammad, "Heuristic approaches to solve traveling salesman problem," 2015.

[7] T. Saradatta and P. Pongchairerks, "A time-dependent ATSP with time window and precedence constraints in air travel," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 2-3, pp. 149–153, 2017.

[8] A. Vyas, D. Chawla, A. Mehta, A. Chelawat, and U. Thakar, "Genetic algorithm for solving the traveling salesman problem using neighbor-based constructive crossover operator," *International Journal of Engineering Sciences and Research Technology (IJESRT)*, vol. 7, pp. 101–110, 2018.

[9] A. Kheiri and E. Keedwell, "A sequence-based selection hyper-heuristic utilising a hidden markov model," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 417–424.

[10] A. Kheiri and E. Özcan, "An iterated multi-stage selection hyper-heuristic," *European Journal of Operational Research*, vol. 250, no. 1, pp. 77–90, 2016.

[11] J.-Y. Potvin, "The traveling salesman problem: A neural network perspective," 1993.

[12] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization - artificial ants as a computational intelligence technique," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 28–39, 2006.

[13] J. Regional Campus, "A review of parameters for improving the performance of particle swarm optimization," *International Journal of Hybrid Information Technology*, vol. 8, no. 4, pp. 7–14, 2015.

[14] O. Mersmann, B. Bischl, J. Bossek, H. Trautmann, M. Wagner, and F. Neumann, "Local search and the traveling salesman problem: A feature-based characterization of problem hardness," in *Learning and Intelligent Optimization*, Y. Hamadi and M. Schoenauer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 115–129.

[15] A. Kheiri and E. Keedwell, "A hidden Markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems," *Evolutionary Computation*, vol. 25, no. 3, pp. 473–501, 2017.

[16] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, 2019.

[17] A. Kheiri, A. G. Dragomir, D. Mueller, J. Gromicho, C. Jagtenberg, and J. J. van Hoorn, "Tackling a VRP challenge to redistribute scarce equipment within time windows using metaheuristic algorithms," *EURO Journal on Transportation and Logistics*, 2019.

[18] D. Wilson, S. Rodrigues, C. Segura, I. Loshchilov, F. Hutter, G. L. Buenfil, A. Kheiri, E. Keedwell, M. Ocampo-Pineda, E. Özcan, S. I. V. Peña, B. Goldman, S. B. Rionda, A. Hernández-Aguirre, K. Veeramachaneni, and S. Cussat-Blanc, "Evolutionary computation for wind farm layout optimization," *Renewable Energy*, vol. 126, pp. 681–691, 2018.

[19] E. K. E. Ahmed, A. M. A. Khalifa, and A. Kheiri, "Evolutionary computation for static traffic light cycle optimisation," in *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2018, pp. 1–6.