generate the k-day itinerary. The algorithm uses a greedy strategy to create an initial solution, which is continuously refined in the adjustment phase. The adjustment phases cans the index to find a potentially better solution.

In the next two sections, we first present how we apply the Map Reduce framework to generate and index the single-day itineraries. The parallel processing engine enables us to search the optimal solution in a brute-force manner. Next, we show after the pre-processing, the complexity of TOP is reduced and approximate algorithms are available.

3. Pre-processing

The pre-processing includes two steps. In the first step, a set of Map Reduce jobs are submitted to produce all possible single-day itineraries. In the second step, the single-day itineraries are reorganized as an itinerary index, which supports efficient itinerary search.

3.1 Intractability of Optimal Itinerary Algorithm

Given a user request (S_p, k), the goal of an itinerary planning algorithm is to provide an itinerary, which ranks highest among all possible itineraries. The score of the itinerary is computed based on the POI weights. However, as shown in the following theorem, this is an NP-complete problem and no polynomial time algorithm exists

Theorem 1.Finding optimal k-day itinerary in a POI graph G= (V, E) is an NP-complete problem.

Proof (Sketch).The optimal k-day itinerary can be reduced to the TOP [3], which is a well-known NP-complete problem. Consider a simple scenario where,

1) K vehicles are created, which start from the same position.
2) Each vehicle has a time limit (1 day) for traveling the POIs.
3) Each vehicle collects the profit by visiting the POIs.
4) The POI accessed by a vehicle will not be considered by other vehicles.
5) The POI's profit is equal to its weight.

The TOP is to find the travelling plan that generates the most profits. The results of the TOP are also the best k-day itinerary.

Due to the complexity of TOP, it is impossible to find the exact solution. Instead, previous work focuses on proposing heuristic algorithms. The basic idea is to generate an initial plan and then adjust it based on some heuristic rules. Those algorithms have three drawbacks. First, the heuristic algorithms need many iterations to get a good enough result, which incur high computation cost [7]. Second, the adjusting rules are too complicated and the potential gains are unknown. Finally, there is no bound of the approximate result, which may be arbitrarily bad in some cases.

In this paper, we reduce the complexity of the TOP by transforming it into a set-packing [8] problem. As the transformation is done in an offline manner, the performance of online query processing is not affected.

3.2 Single-Day Itinerary

The basic idea of transformation is to iterate all possible single-day itineraries. This is done by a set of Map Reduce jobs. In the first job, we generate |P| initial itineraries for the POI set P. Each initial itinerary only consists of one POI. Iteratively, the subsequent Map Reduce job tries to add one more POI to the itineraries. If no more single-day itineraries can be generated, the process terminates. In current implementation, we allow maximally m Map Reduce jobs in the transformation process to reduce the over heads. Therefore, a single-day itinerary contains at most m POIs. This strategy is based on the assumption that users cannot visit too many POIs in one day. In our crawled data set from Yahoo travel, setting m to 10 is enough for Singapore data, which include more than 400 POIs. Only a few single-day itineraries can contain more than 10 POIs.

Algorithms 1 and 2 show the pseudo codes of the Map Reduce job. Them appears load the partial paths from the DFS, which are generated in the previous Map Reduce jobs. We try to append new POI to the existing itineraries. For each new path, we test whether it can be completed within one day. If not, we will discard the new path. If the old path cannot result in any new path, we will output the old path. For the last Map Reduce job (them th job), all the candidate itineraries are used as the results. The output key-value pair is using the sorted POIs in the itinerary as the key.

```
Algorithm 1. map(Object key, Text value,
Context context).
// we allow maximally m – round MapReduce jobs, i.e.,
the maximally length of path is m
//value: existing path, each MapReduce job tries to add one
more POI to the path
Path P = parsePath(value)
for i = 0 to POIGraph.POINumber do
    if isConnected(P, i) and !P.contains(i) then
        Path newPath = P.append(i)
        cost = P.cost + POIGraph.getCost(P.endPOI, i)
        + POIGraph.getCost(i)
        weight = P.weight + POIGraph.getWeight(i)
        newPath.cost = cost
        newPath.weight = weight
        if newPath.cost ≤ H then
            Key newKey = parsePath(newPath).sort();
            context.collect(newKey, newPath)
        else
            DFS.write(resultFile, P)
```

```
Algorithm 2. reduce(Key key, Iterable values,
Context context).
bestCost = ∞
bestPath = NIL
for Path P: values do
    if P.cost < bestCost then
        bestPath = P
        bestCost = P.cost
context.collect(key, bestPath)
```

In the mappers, to compute the weight and cost of new itinerary, we load the POI graph table from the DFS. As the graph table is small, each reducer maintains a copy in its memory. The table's schema is as follows:

$(S\_POI, E\_POI, S\_weight, E\_weight, S\_cost, E\_cost, cost)$,

Where S_POI and E_POI denote the two POIs linked by aspecific edge,costis the traveling cost from S_POI to E_POI,and S_POI is the primary key of the table.

In the reducers (Algorithm 2), we select the path with smallest cost of paths with the same POIs. In each reducer, all the paths have the same POIs. We only keep the path with smallest cost and output such path for the next round.Note that since all the paths have the same POIs, these paths have the same weight.

After all itineraries have been generated, a clean process is invoked to remove the duplication. For $(L_0 = v_0 \rightarrow \cdots \rightarrow v_n$ and $L_1 = v'_0 \rightarrow \cdots \rightarrow v'_n)$, L_0 contains L_1, if

$$\forall v'_j \in L_1 \rightarrow \exists v_i \in L_0 (v_i = v'_j).$$

Namely, all POIs inL1 are also included by L0. IfL0 Contains L1, we will only keep L0, as it provides more POIs for the users.

3.3 Itinerary Index

To efficiently locate the single-day itineraries, an inverted index is built. The key is the POI and the values are all itineraries involving the POI. By scanning the index, we can retrieve all the itineraries. We create an index file for each POI in the DFS. The file includes all single itineraries involving the POI, which are sorted based on their weights. For example, in Fig. 4, "1.idx" contains all itineraries for the first POI. The itinerary "1|5|20|12|40" is the most important itinerary in the index file with weight 320.

The inverted index is constructed via a Map Reduce job Algorithms 3 and 4 show the process. The mappers load the single-day itinerary and generate key-value pairs for each involved POI. The reducers collect all itineraries for a specific POI and sort them based on the weights before creating the index file. In our system, the size of the index file may vary alot. Some POI may have an extremely large index file, due to its popularity and short visit time. In reducers, those POIs may result in the exception of memory overflow in the sorting process. To address this problem, in the map phase, instead of using the POI as the key, we generate the composite key by combining the POI and the itinerary weight.

Algorithm 3. `map(Object` *key*, `Text` *value*, `Context` *context*).

```
//value: single-day itinerary
Itinerary it = parse(value)
for i = 0 to it.POISize() do
    int nextPOI = it.getNext(i)
    Key key = new CompositeKey(nextPOI, it.weight/
    bucketSize)
    context.collect(key, it)
```

Algorithm 4. `reduce(Key` *key*, `Iterable` *values*, `Context` *context*).

```
CompositeKey ck = key, Set s = ∅
for Itinerary it: values do
    s.add(it)
sort(s)
DFSFile f = new DFSFile(ck.first + "_"+ck.second)
f.write(s)
```

In particular, we partition the itineraries into n buckets. The bucket ID is used as a part of the composite key. In this way, we split the itineraries of a POI into n groups and each group can be efficiently sorted in the memory. Each group will result in an index file. However, it is not necessary to merge the files, as the files are partitioned based on the weights. By scanning all files from the nth bucket to the 1th bucket, we can get a sorted list for all itineraries involving POI.

To simplify the index manipulation, an index manager is built in our query engine. The index manager only provides one interfaces can (POI), where POI denotes the owner of the index. The interface returns an iterator, which can be used to retrieve all itineraries of the POI. A memory buffer is established to cache the used itineraries and the LRU strategy is applied to maintain the buffer.

3.4 Discussion: Why Map Reduce

Although the input data set (POI graph) is small in size, the partial results of the possible itineraries are extremely large (more than 100G or even 1T). The computation is also intensive, which cannot be completed by a single machine. MapReduce is the solution to partition the partial results and generate the itineraries in parallel. Its advantages are twofold:

1) Parallel computing effectively reduces the running time of pre-processing. The search space explodes, when the number of POIs and traveling days increases. It is impractical to generate all possible itineraries. But by exploiting the power of Map Reduce, we can share and balance the workload between multiple machines. The scalability is achieved by adding more nodes into the cluster. In our experiment, the running time of pre-processing is significantly reduced with the number of nodes.
2) Map Reduce algorithms can remove the duplicated itineraries in a simple way. In Algorithm 2, by leveraging the framework of Map Reduce, we map all the itineraries with the same POIs into the same reducer and only keep one itinerary with the lowest cost. This approach can prune the low-benefit partial itineraries as early as possible and lead to less input for the next round of computation.

4. Related Work

Most existing work on itinerary generation take a two-step scheme. They first adopt the data mining algorithms to discover the users' traveling patterns from their published images, geolocations and events [11], [12], [13]. Based on the relationships of those historical data, new itineraries are generated and recommended to the users [14], [15], [16].This scheme leverages the user data to retrieve POIs and organize the POIs into itinerary, which is based on a different application scenario to ours. We help the traveling agency provide the customized itinerary service, where all details of POIs are known and each user prefers different itinerary instead of adopting the most popular ones. In our case, the itinerary generation problem is a search problem for the optimal POI combinations.
In fact, searching for the optimal single-day itinerary has been well studied. It can be transformed into the traveling salesman problem (TSP) [5], which is a well-known NP-complete problem. For example, in [17], given a set of POIs,the system will generate a shortest itinerary to access all the POIs. If the distance measure is a metric and symmetric, the TSP has the polynomial approximate solution [18], but the approximate solution incurs high overhead for a large POI graph [19]. Therefore, some heuristic approaches [1] are adopted to simplify the computation.

Some interactive search algorithms [2], [20] are proposed in recent years. These algorithms still focus on optimal single-day itinerary planning. To reduce the computation overhead and improve the quality of generated itineraries, users' feedbacks are integrated into the search algorithm. The search algorithm works iteratively. It proposes new itineraries for users based on their previous feedbacks and the users can adjust the weights of POIs in the itinerary or select new POIs into the itinerary. In the next iteration, the algorithm will refine its results based on the collected information. Those works can be considered as variants of optimal single-day itinerary planning problems, where as our algorithms focus on generating multi-day itineraries. Moreover, interactive algorithms pose requirements for the users, who may be reluctant to provide the feedbacks.

To the best of our knowledge, no previous work studied the problem of generating multiday itinerary. This problem is more challenging than the single-day itinerary, because simply combining multiple optimal single-day itineraries may result in a suboptimal solution. The multiday itinerary, as shown in this paper, can be reduced to the team orienting problem (TOP) [3], which is an NP-complete problem with no approximate solution. Therefore, many heuristic approaches are proposed [6], [21], [22]. The heuristic approaches cannot guarantee the quality of generated itineraries. The weight ratio is computed between the MR-Set with adjustment and MR-Set without adjustment. the Map Reduce framework to generate the single-day itineraries. The parallel engine of Map Reduce allows us to solve some NP-complete problems more efficiently. Other work [23], [24] also try to leverage the power of Map Reduce to reduce the processing cost of NP-complete problems. The beauty of our approach is that after the transformation, the itinerary planning problem is reduced to the weighted set-packing problem, which has approximate solutions under some constraints.

5. Conclusion

In this paper, we present an automatic itinerary generation service for the backpack travellers. The service creates a customized multiday itinerary based on the user's preference. This problem is a famous NP-complete problem, team orienting problem, which has no polynomial time approximate algorithm. To search for the optimal solution, a two-stage scheme is adopted. In the pre-processing stage, we iterate and index the candidate single-day itineraries using the Map Reduce framework. The parallel processing engine allows us to scan the whole dataset and index as many itineraries as possible. After the pre-processing stage, the TOP is transformed into the weighted set-packing problem, which has efficient approximate algorithms. In the next stage, we simulate the approximate algorithm for the set-packing problem. The algorithm follows the initialization-adjustment model and can generate a result, which is at most $\frac{2(m+1)}{3}$ worse than the optimal result. Experiments on real data set from Yahoo's traveling website show that our proposed approach can efficiently generate high-quality customized itineraries.

References

[1] S. Dunstall, M.E. Horn, P. Kilby, M. Krishnamoorthy, B. Owens,D. Sier, and S. Thiebaux, "An Automated Itinerary Planning System for Holiday Travel,"Information Technology and Tourism,vol. 6, no. 3, pp. 195-210, 2004.
[2] S.B. Roy, G. Das, S. Amer-Yahia, and C. Yu, "Interactive ItineraryPlanning,"Proc. IEEE 27th Int'l Conf. Data Eng. (ICDE),pp. 15-26,2011.
[3] I.-M. Chao, B.L. Golden, and E.A. Wasil, "The Team OrienteeringProblem,"European J. Operational Research,vol. 88, no. 3, pp. 464-474, Feb. 1996.
[4] J. Dean and S. Ghemawat, "MapReduce: A Flexible DataProcessing Tool,"Com
[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein,Introductionto Algorithms, second ed. The MIT Press and McGraw-Hill BookCompany, 2001.
[6] C. Archetti, A. Hertz, and M.G. Speranza, "Metaheuristics for theTeam Orienteering Problem,"J. Heuristics,vol. 13, pp. 49-76, Feb.2007.
[7] P. Vansteenwegen, W. Souffriau, and D.V. Oudheusden, "TheOrienteering Problem: A Survey,"European J. Operational Research, vol. 209, pp. 1-10, Feb. 2011.
[8] M.M. Halldo´rsson and B. Chandra, "Greedy Local Improvementand Weighted Set Packing Approximation,"J. Algorithms,vol. 39,pp. 223-240, May 2001.
[9] E.M. Arkin and R. Hassin, "On Local Search for Weighted K-SetPacking,"Math. Operations Research,vol. 23, pp. 640-648, Mar.1998.
[10] http://hadoop.apache.org/, 2013.
[11] T. Rattenbury, N. Good, and M. Naaman, "Toward AutomaticExtraction of Event and Place Semantics from Flickr Tags,"Proc.30th Ann. Int'l ACM SIGIR Conf.

Research and Development in Information Retrieval (SIGIR '07),pp. 103-110, 2007.
[12] D.J. Crandall, L. Backstrom, D.P. Huttenlocher, and J.M.Kleinberg, "Mapping the World's Photos," Proc. 18th Int'lConf. World Wide Web (WWW),pp. 761-770, 2009.
[13] M. Clements, P. Serdyukov, A.P. de Vries, and M.J. Reinders,"Using Flickr Geotags to Predict User Travel Behaviour,"Proc.33rd Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR),2010.
[14] C.-H. Tai, D.-N. Yang, L.-T. Lin, and M.-S. Chen, "RecommendingPersonalized Scenic Itinerary with Geo-Tagged Photos,"Proc.IEEE Int'l Conf. Multimedia and Expo (ICME),pp. 1209-1212, 2008.
[15] M.D. Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R.Lempel, and C. Yu, "Automatic Construction of Travel ItinerariesUsing Social Breadcrumbs,"Proc. 21st ACM Conf. Hypertext and Hypermedia (HT),pp. 35-44, 2010.
[16] H. Yoon, Y. Zheng, X. Xie, and W. Woo, "Smart ItineraryRecommendation Based on User-Generated GPS Trajectories,"Proc. Seventh Int'l Conf. Ubiquitous Intelligence and Computing (UIC),pp. 19-34, 2010.
[17] I. Hefez, Y. Kanza, and R. Levin, "TARSIUS: A System for Traffic-Aware Route Search under Conditions of Uncertainty,"Proc. 19thACM SIGSPATIAL Int'l Conf. Advances in Geographic InformationSystems (GIS),pp. 517-520, 2011.
[18] N. Christofides, "Worst-Case Analysis of a New Heuristic for theTraveling Salesman Problem," Technical Report 388, GraduateSchool of Industrial Administration, Carnegie-Mellon Univ., 1976.
[19] G. Laporte, "The Traveling Salesman Problem: An Overview ofExact and Approximate Algorithms," European J. OperationalResearch,vol. 59, no. 2, pp. 231-247, June 1992.
[20] R. Levin, Y. Kanza, E. Safra, and Y. Sagiv, "Interactive RouteSearch in thePresenceofOrder Constraints," Proc. VLDBEndowment,vol. 3, no. 1, pp. 117-128, 2010.
[21] W. Souffriau, P. Vansteenwegen, G.V. Berghe, and D.V.Oudheusden, "A Path Relinking Approach for the TeamOrienteering Problem," Computers and Operations Research,vol. 37, pp. 1853-1859, 2010.
[22] M.V.S.P. de Aragao, H. Viana, and E. Uchoa, "The TeamOrienteering Problem: Formulations and Branch-Cut and Price,"Proc. Algorithmic Approaches for Transportation Modeling, Optimiza-tion, and Systems (ATMOS), vol. 14, pp. 142-155, 2010.
[23] F. Chierichetti, R. Kumar, and A. Tomkins, "Max-Cover in Map-Reduce,"Proc. 19th Int'l Conf. World Wide Web (WWW),pp. 231-240, 2010.
[24] Z. Zhao, G. Wang, A.R. Butt, M. Khan, V.A. Kumar, and M.V.Marathe, "SAHAD: Subgraph Analysis in Massive NetworksUsing Hadoop,"IEEE Int'l Parallel and Distributed Processing Symp.(IPDPS),2012

Author Profile

Bonuguntla Saranya obtained degree in B. Sc computers from Ahyudaya Mahila College, Guntur and obtained MCA in Acharya Nagarjuna University, Guntur. At present persuing M. Tech in computer science and engineering in Guntur Engineering college, Guntur.

Miryala Venkatesh obtained B. Tech in Computer science and Engineering in Narasaraopet Engineering College in 2003.MS in latrob university, Australia in 2005, He has Industrial experience for 3Yrs and teaching experience for 7yrs and presently working at Guntur Engineering College, Guntur.