Comparison of Search Algorithms on Flight Search

Zhenyang Pan, Wenshuai Ye, Xide Xia

Harvard University
School of Engineering and Applied Science

Abstract

Travel is the movement of people between relatively distant geographical locations. Prior to the trip, one needs to come up with a plan on destinations, air tickets, visa, and many other issues. Rather than come up with a plan manually, we can ask our computers for help. With the help of artificial intelligence, we developed a system named FlightExpert, which selects the cheapest airfare ticket given certain constraints by the user. Constraints include the maximum number of stops, arrival time, etc. The algorithms and concepts covered include IDA search, beam search, stochastic beam search, bi-directional search, and using regression to learn a heuristic function. Of all the search algorithms we have implemented, bi-directional search and IDA* search with an admissible heuristic function can find optimal solutions. In addition, bi-directional search and local search are less time consuming compared with IDA* search.*

Connecting flights are pretty common in the US nowadays. Whether people fly domestically or internationally, they are likely to stop at transition airports before reaching their final destinations if they want to travel at the lowest cost due to the sheer number of regional airports supply for connecting flights here. However, this is not the case in Asia in that regional airports are not equally well developed there. When passengers get on a flight, their options have been limited to direct flights over the past years. People believe flights with stops come with greater cost. Nevertheless, with the rapid development of Asia economy, more and more regional airports are actually being built. As a result, searching for connecting flights to find the cheapest airfare ticket becomes feasible and reasonable. In this paper, we explore various search algorithms by applying them to the flight data scraped from Kayak[1]. Specifically, we want to find a path between two airports using efficient search algorithms. Based on these criteria, we have selected IDA* search, beam search, stochastic beam search, and bi-directional search with different heuristic functions. The first section of this paper discusses how we formalize the problem. Section 2 gives an introduction to the application of the algorithms, whereas section 3 deals with comparison of various search algorithms. The last section draws a conclusion of our analysis.

[1] www.kayak.com

I. Problem Formalization

Before performing analysis and implemented the code, we need to formalize the problem first. We scraped the data from Kayak using their public API. Because they imposed a restriction to the amount of data we could get, we only retrieved a single month of data in Asia. The data contains information on departure time, arrival time, departure airport, arrival airport, price of the flight, flight class, flight number, and the airline. We have saved each type of information mentioned in a separate text file. Below is the formalization of our flight search problem.

- State: $(Location, ArrivalTime)$,
- Initial State: $(DepartureAirport, Time_0)$,

- Goal State: $(ArrivalAirport, Time_1)$,
- Action: *Flight*
- Cost: *FlightTicketPrice*

Location above constituting a state refers to the airport, whereas the arrival time is the time one arrives at the airport. The next flight must take off after this time. In the goal state, this time component refers to earliest time the travellers need to get to their final destinations. Currently the program is hard coded to ensure that the flight should arrive within 2 days of this earliest time because lower tolerance might prevent the our search algorithms from finding any flight due to the limited availability of the Kayak data. With states clearly defined, it is easy to dig out the flight that connects two states. In the informed search algorithms, we need to come up with a heuristic function. A simple and admissible heuristic here is the lowest possible price from the current airport, regardless where the flight is going. Since it will always underestimates the real cost from the current state to the goal state, this heuristic function is admissible and will help us find an optimal solution. We will discuss how we can get a better heuristic using regression technique in the later section of this paper.

II. Algorithms Exploration

In this section, we will explore some classic search algorithms applied to flight ticket search. The algorithms involve both global search and local search. We will discuss the trade-off between various algorithms.

II.1 IDA* Search

The state is a combination of airport and time. As a result, there will be many successors with regards to an airport, implying that the space usage is very large in the search algorithms. Since the search space is quite large, having a priority queue to implement A* search to store all the states may not be efficient. As an alternative A*, IDA* turns out to be a relatively good choice in this situation that will guarantee an optimal solution.

IDA* is a variant of iterative deepening depth-first (IDS) search that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the A* search algorithm. Each iteration is a depth-first search where a branch is pruned when it reaches a node whose total cost exceeds the cost cutoff value of that iteration. The cost threshold for the first iteration is the heuristic value of the initial state, and increases in each iteration to the lowest cost of all nodes pruned on the previous iteration. It continues until a goal node is found whose cost does not exceed the current cost threshold. Since it is based on iterative deepening depth-first search, its memory usage is lower than in A*, which can be desirable in our case. The drawback, however, is that IDA* consumes more time than A* search.

Improving our heuristic Although our heuristic function, which chooses the lowest price possible from the current state, is admissible and hence guarantees an optimal solution, it underestimates the true cost dramatically. To tackle this problem, we borrow the idea from Ernandes and Gori (2004). They call their method "bootstrap learning of heuristic functions". The basic idea is that initially we solve some instances of our problem using our weak heuristic function h_0. We call these instances bootstrap instances. By solving n instances, we will have a collection of size n samples. These samples, of which we have already known the true cost, is then fed to a learning algorithm to create a new heuristic function. In our case, we have skipped the bootstrap resampling part and focused only on fitting a linear regression using the direct flight duration between two airports as the only feature. Such feature is based on the assumption that a direct flight does exist between two airports. It is irrefutable that such learning algorithm does not guarantee the heuristic function is optimal. However, as n increases, our solution will be very close to optimal.

Figure 1: Regression Summary

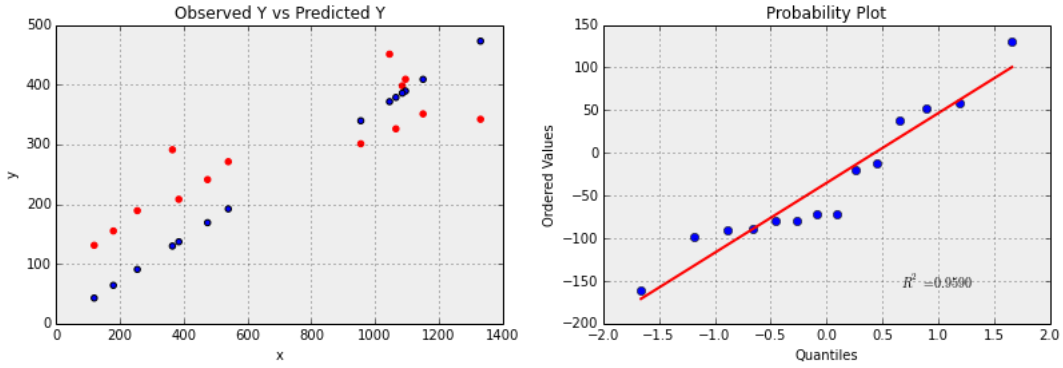
OLS Regression Results

Dep. Variable:	optimal	**R-squared:**	0.923
Model:	OLS	**Adj. R-squared:**	0.917
Method:	Least Squares	**F-statistic:**	156.6
Date:	Sat, 06 Dec 2014	**Prob (F-statistic):**	1.26e-08
Time:	20:02:01	**Log-Likelihood:**	-81.971
No. Observations:	14	**AIC:**	165.9
Df Residuals:	13	**BIC:**	166.6
Df Model:	1		

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.770
Model:	OLS	**Adj. R-squared:**	0.750
Method:	Least Squares	**F-statistic:**	40.10
Date:	Wed, 10 Dec 2014	**Prob (F-statistic):**	3.76e-05
Time:	13:57:45	**Log-Likelihood:**	-73.141
No. Observations:	14	**AIC:**	150.3
Df Residuals:	12	**BIC:**	151.6
Df Model:	1		

Figure 2: Regression Analysis Plot



We tested this method with 14 instances collected. Figure 1 shows the regression output. The one on the left corresponds to the regression model without including an intercept, whereas the figure on the right refers to the model with an intercept. The model without the intercept shows that 92.3% of the data can be explained by the model. This R square value exceeds the one from the model with the intercept. As a result, we selected the first model for our heuristic function. In addition, the qq plot shown in 2 further verifies the reliability of our model by displaying that the residuals are normally distributed in our case, indicating that the regression assumption hold here.

II.2 Beam Search

To further reduce the time cost, we now switch to local search algorithms to search the flights. Beam search turns out to be a good candidate in this scenario because it can keep track of k successors, which in a large sense prevents the algorithm from getting stuck in a point. However, the traditional beam search ignores the path and keeps information on states only. We customize the beam search algorithm by using a dictionary to keep track of the parent of each successor state.

Specifically, we first choose k random successors (k = 10 in our testing) from the initial state. For each successor, we again select all of its next successors to be the candidates of our next states. Then we pick k best states as our next states. "Best" is defined as the lowest cumulative price from the initial state to the states in the candidate list. When a solution is found, the program will backtrack the dictionary to get the path.

The advantage of beam search here lies in

the fact that it consumes less time than IDA* without increasing the memory usage a lot. However, the disadvantage is obvious. Just as other local search does, beam search cannot guarantee an optimal solution.

II.3 Stochastic Beam Search

Similar to beam search, stochastic beam search keeps track of k successors to prevent the algorithm from getting stuck in a point. The difference is that stochastic beam search generates successor states from a distribution biased towards better states, which further enhances the chance of finding a solution. Inspired by the idea that a flight with lower price tends to be "better", the successor distribution is an empirical distribution biased towards successor states with lower cumulative price. The step is listed below.

1 Create a successor list containing all the successors of the current k states.
2 Aggregate the cumulative price of all the successor states, denoted as P.
3 For each state s, use the aggregate price divided by its own cumulative price p_s and multiply a factor m, a constant to control the space of the list.
4 Generate a new list for each successor state. Each state should appear $\frac{Pm}{p_s}$ in this list. If $\frac{Pm}{p_s}$ is less than 1, we round it up to 1.
5 Iteratively draw states from this newly constructed list and store the selected samples in a new list. Once a state has been drawn, remove it from the empirical distribution list.
6 Return the newly constructed list that contains k sample states as successors.

Again, we need to keep track of the path in stochastic beam search. Compared with beam search, it is less likely to get stuck in the middle at the expense of using extra space storage.

II.4 Bidirectional Search

To further increase time efficient without losing optimality, we shift our focus to bidirectional search, which runs two A* search algorithms simultaneously. We search from two directions, one from the initial state and the other from the goal state defined in our problem. To do the backward search from the goal state, we create a function to get predecessors that can return the states before arriving at the current state, a function conceptually similar to generating successors with different direction. In addition, we keep two priority queue to track all the lowest price states from the begin and from the end. That is, search algorithms from both starting point are guided by a heuristic estimate of the remaining distance to the goal. Once there is an intersection between the visited airport from the begin and end, then we can conclude that an optimal solution has been found. Our results have shown it not only consumes less time than A* but also guarantees the optimal solution. There are cases where a bidirectional search can lead to substantial savings. For example, if the forward and backward branching factors of the search space are both b, and the goal is at depth k, then A* search will take time proportional to bk, whereas a symmetric bidirectional search will take time proportional to $\frac{2bk}{2}$. This is an exponential savings in time compared with our original A* search algorithm.

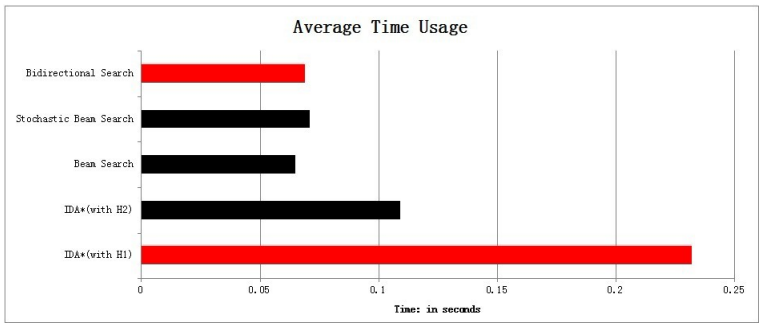
III. Algorithm Comparison

In this section, we discuss the actual performance of different algorithms in our experiment.

Optimality and Completeness IDA* with the admissible heuristic is optimal. But the regression heuristic could make it not optimal because it could lead to overestimate of the true cost. The local search algorithms, namely, beam search and stochastic beam search, are suboptimal due to the randomness involved in them. In contrast, the bidirectional search can find the optimal solution.

In terms of completeness, IDA* and Bidirectional Search are complete. While other local search algorithms do not guarantee completeness.

Figure 3: Time Usage

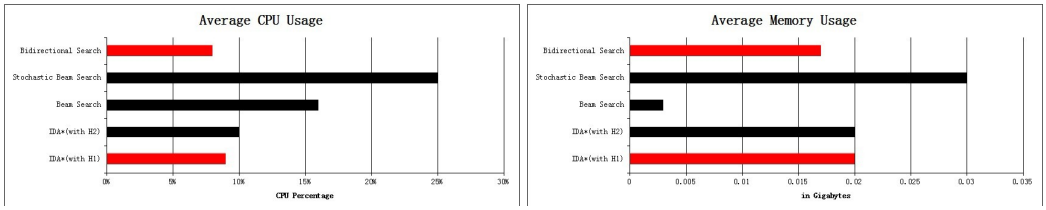


Red color implies that the algorithm can guarantee optimality.

Time Complexity From the average time usage plot shown in 3, we can observe that Beam Search uses the least amount of time. Bidirectional Search and Stochastic Beam Search consume a bit more time than Beam Search. IDA* with the regression model as heuristic is more time efficient than the IDA* with the simple heuristic, though the heuristic function is not admissible.

Space Complexity From 4, we find that Beam Search uses the least amount of memory compared to other search algorithms. Stochastic search consumes heavy memory because it generates an empirical distribution for each successor list. Surprisingly, bidirectional search uses a bit less memory than IDA*. Given a cut-off value, IDA* search will restart the search if there is no solution found. As a result, it consumes less space than local search algorithms we have explored. However, based on the graph, it uses more memory than bidirectional search.

Figure 4: CPU and Memory Usage



Red color implies that the algorithm can guarantee optimality.

IV. Conclusion and Future Work

Our research has revealed that low cost transit flights do exist to connect two large international airports in Asia. Having incorporating knowledge from artificial intelligence and machine learning, we managed to develop a search system using IDA* search, beam search, stochastic beam search, and bidirectional search. In our sample test, IDA* and bi-directional search algorithms with admissible heuristic functions give us some amazingly low fares, and our database has proven that those are the cheapest tickets between the two airports. Moreover, bidirectional search, which produces less space and time, turns out to be a superior algorithm to IDA* in our application. Beam search and stochastic beam search, how-

ever, do not guarantee optimality, though they are relatively time efficient. We later explored some more complicated heuristic function by applying learning algorithms to our sample data. This heuristic is suboptimal. However, as the sample size increases, a heuristic close to optimality can be returned.

For the sake of our own interest, our next step is to incorporate our search algorithms into a user interface. For example, we can provide a GUI for the flight search system, which the user has the option to choose a search algorithm for their flight search. If the user is known for his/her impatience, local search algorithms will be adopted. Otherwise, bidirectional search or IDA* are great.

In terms of data, we have not retrieved all the flight data available from Kayak API due to their restrictions. In our next step, we can spend more time scraping the flight data from Kayak or other flight information database to make an even better comparison of our algorithms.

Lastly, the heuristic function we have constructed adopts a simple regression model and does not guarantee admissibility. We can keep working on this by incorporating more data, such as the direct distance from the departure airport to the arrival airport in our heuristic model. With a better heuristic, the informed search will be more efficient and closer to optimality eventually.

References

[1] Ernandes, M., and Gori, M. 2004. Likely-admissible and sub-symbolic heuristics. *ECAI* 613ÍC617.

[2] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach (2nd ed.)*, Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2.

[3] Korf, Richard E. "Depth-first iterative-deepening: An optimal admissible tree search." Artificial intelligence 27.1 (1985): 97-109.

[4] Koll, Andreas L., and Hermann Kaindl. "Bidirectional best-first search with bounded error: Summary of results." *Proceedings of the 13th international joint conference on Artifical intelligence*-Volume 1. Morgan Kaufmann Publishers Inc., 1993.

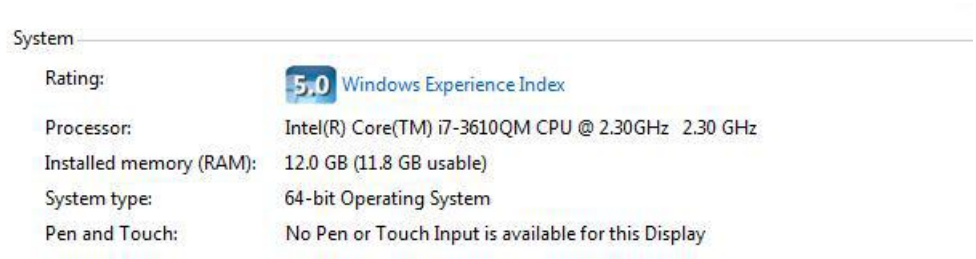
[5] Korf, Richard E., Michael Reid, and Stefan Edelkamp. "Time complexity of iterative-deepening-A*." *Artificial Intelligence* 129.1 (2001): 199-218.

[6] Nannicini, Giacomo, et al. "Bidirectional A* search for time-dependent fast paths." *Experimental Algorithms* Springer Berlin Heidelberg, 2008. 334-346.

A. Appendix I: Demonstration of Our Program and Sample Output

Our equipment is

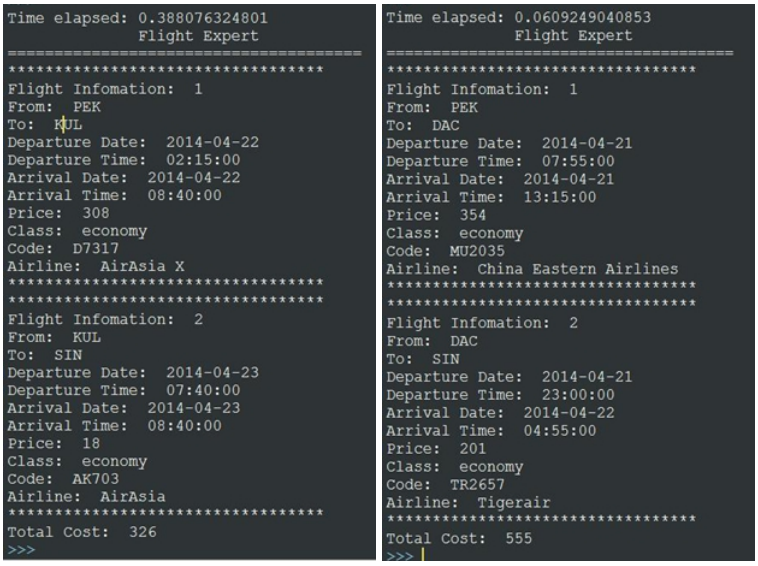
Figure 5: Laptop



Our python input is:
$start = (\text{"}PEK\text{"}, hash_time(20, 12, 0), 0, -1)$
$goal = (\text{"}SIN\text{"}, hash_time(21, 24, 0), 0, -1)$
The input means that the user is looking for a flight from Beijing International Airport (PEK) to Singapore Changi Airport (SIN) after 12pm, April 20. And he should reach Singapore no later than 2 days after after 0am, April 22. The "2 days" here is another parameter that we can change in our program.

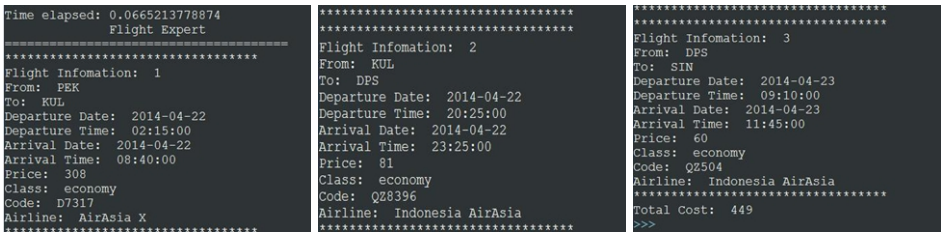
IDA* search with h_0 as heuristic function result and beam search demo

Figure 6: IDA Search and Local Beam Search Result*



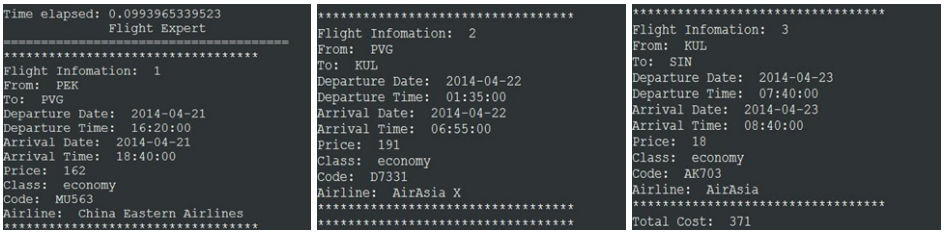
stochastic beam search demo

Figure 7: Stochastic Beam Search



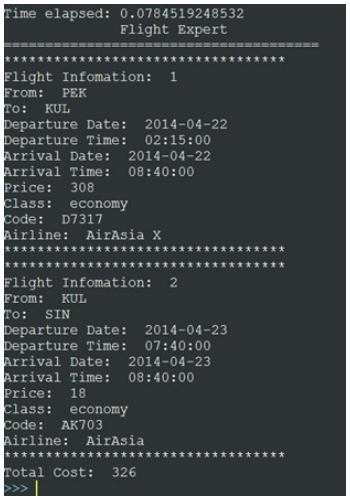
IDA* search with h_1 as heuristic function result demo.

Figure 8: IDA Search with Regression as A Heuristic Function*



bidirectional search demo

Figure 9: Bidirectional Search



A. Appendix II: Code Execution

First, make sure the code and the data are in the same folder. Our original data is saved in the inst. file called flight. We used parser to convert it to several txt files. After that, Simply open the algorithm file and click run. The default parameter of the search is
$start = (\text{"}PEK\text{"}, hash_time(20, 12, 0), 0, -1)$
$goal = (\text{"}SIN\text{"}, hash_time(21, 24, 0), 0, -1)$

If you want to search for tickets for another route, all you need to do is to change the first two elements in the tuples. For example, if you want to find the ticket from Shanghai to Singapore without changing the time, you can change start to $(\text{"}PVG\text{"}, hash_time(20, 12, 0), 0, -1)$.

These parameters are located in the following lines respectively.

IDA* Search with h_0: line 68-69
IDA* Search with h_1: line 59-60
Beam Search: line 50-51
Stochastic Beam Search: line 52-53
Bidirectional Search: line 71-72

To view the regression process for h_1, make sure ipython notebook has been installed and configured.

A. Appendix III: Participants and Contributions

Task	*Wenshuai*	*Zhenyang*	*Xide*
Gathered flight data	✓	✓	✓
Converted data into txt files			✓
Implemented IDA* search	✓	✓	✓
Developed heuristic functions	✓		
Implemented beam search	✓	✓	✓
Implemented stochastic beam search	✓	✓	✓
implemented bidirectional search		✓	
Performed system testing		✓	
Made presentation slides	✓	✓	✓
Wrote final report	✓	✓	✓
Compiled in latex	✓		