Project 4: Dynamic vs. Exhaustive

Dennis Newman dlnew3@csu.fullerton.edu

Rosa Cho rkcho317@csu.fullerton.edu

Mathematical Analyses

Dynamic Programming Algorithm

1. Analyze your dynamic programming algorithm code mathematically to determine its big-O efficiency class, probably $O(n^2)$ or $O(n \log n)$.

std::unique_ptr<RideVector> dynamic_max_time
(
const RideVector& rides,
int total_cost
)
{
std::unique_ptr<RideVector> output(new RideVector);
 	
std::vector<double> vecTime;
std::vector<int> vecCost;
for (int i = 0; i < rides.size(); i++){
vecTime.push_back(rides[i]->time());
vecCost.push_back(rides[i]->cost());
}
int cache_x = rides.size();
std::vector<std::vector<int>> cache(cache_x + 1, std::vector<int>(total_cost + 1));
for (int x = 0; x <= cache_x; x++){
for (int y = 0; y <= total_cost; y++){
if (x == 0 || y == 0){
cache[x][y] = 0;
}
else if (vecCost[x - 1] <= y){
cache[x][y] = max(vecTime[x - 1] + cache[x - 1][y - vecCost[x - 1]], cache[x - 1][y]);
}
else{ cache[x][y] = cache[x - 1][y]; }
}
}
int cache_result = cache[cache_x][total_cost];
int cost = total_cost;
for (int z = cache_x; z > 0 && cache_result > 0; z--){
if (cache_result == cache[z - 1][cost]){
continue; }
	3 + (N *
	1
	1
	) = 3 + 2N
	1
	2
	
	3 + (N *
	3 + (M *
	max(3
	1
	),
	(2
	8),
	
	(2)
	) = 3 + (N * (3 + (M * 10)))
	
	1
	1
	5 + (N
	max(2
	0),

else{ cache_result -= vecTime[z - 1];
cost -= vecCost[z - 1];
output->push_back(rides[z - 1]); }
}
return output;
}
	(2
	2
	1)
	) = 5 + N(5)
	1

Conclusion:	Total Number of steps = 3 + 2N + 1 + 2 + 3 + N(3 + M(10)) + 1 + 1 + 5 + N(5) + 1 = 10N + 10NM + 17 = 10N(1+M) + 17 = O(n^2)

Exhaustive Optimization Algorithm

2. Analyze your exhaustive optimization algorithm code mathematically to determine its big-O efficiency class, probably $O(2^n \cdot n)$.

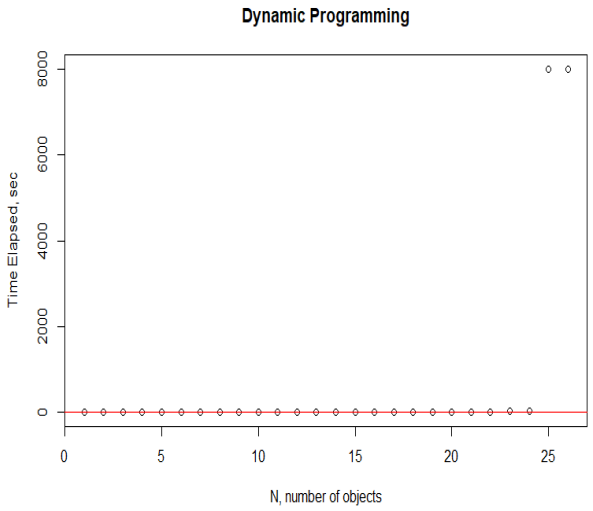
std::unique_ptr<RideVector> exhaustive_max_time (const RideVector& rides,	

double total_cost)
{
std::unique_ptr<RideVector> output(new RideVector);
RideVector best;
double best_cost = 0;
double best_time = 0;
unsigned int bit_length = pow(2, rides.size());
for (int i = 0; i < bit_length; i++){
RideVector subsets;
double subset_cost = 0;
double subset_time = 0;
for (int j = 0; j < rides.size(); ++j) {
if ((i >> j) & 1){
subsets.push_back(rides[j]);
subset_cost += rides[j]->cost();
subset_time += rides[j]->time();
}
}
if (subset_cost <= total_cost){
if (best.empty() || (subset_time > best_time)){
best = std::move(subsets);
best_cost = subset_cost;
best_time = subset_time;
}
}
subsets.clear();
}
for (int i = 0; i < best.size(); ++i){
output->push_back(best[i]);
}
return output;
}
	1
	1
	2
	3 * 2^N-1(
	
	1
	1
	3 + N(
	2
	1
	1
	1
	
	) = 3 + N(5)
	1
	2
	1
	1
	1
	
	) = 3* (2^N - 1)(5 + N(5) + 6)
	3 + N(
	1
	)
	1
	
	= 7 * (2^N-1)(5 + N(5) + 6) + 3 + N + 1
	=7 * (2^N-1)(11 + 5N) + N +4

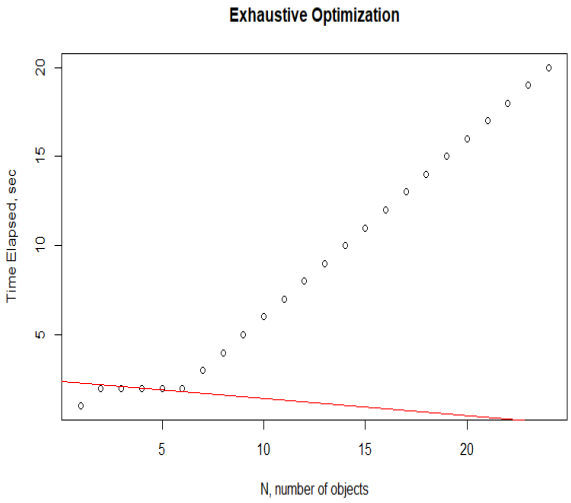
Conclusion:	Total Number of steps = $O(2^n * n)$

Scatterplots

DYNAMIC PROGRAMMING



EXHAUSTIVE OPTIMIZATION



3. <u>Conclude whether or not your empirically-observed time efficiency data is consistent, or inconsistent, with your mathematically-derived big-*O* efficiency class for each algorithm.</u>
   As shown by these scatterplots, the time efficiency of each algorithm is visibly different. Dynamic Programming as a consistently constant slope as the number of *n* increases with the exception of the outliers at the very end. Its best-fit line has an almost straight slope from beginning to end. Therefore, its time efficiency reflects a constant O(1) time which is quite different from our mathematical analysis of $O(n^2)$. On the other hand, Exhaustive Optimization shows a clear, gradually increasing slope as the number of *n* increases; this shows a positive relationship between time elapsed (in seconds) and the number of *n* in this algorithm. However, its best-fit line is a negative slope going in the opposing direction of the dataset's own. Therefore, its time efficiency reflects an O(n log n) time which, once again, is very different from the mathematical analysis which we concluded as $O(2^n * n)$. Our conclusion is that our empirically-observed time efficiency data is inconsistent with our mathematically-derived big-O efficiency class.

Hypotheses Examination and Analysis

4. Answers to the following questions, using complete sentences.
    a. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?

       The Dynamic Programming Algorithm is substantially faster than the Exhaustive Search Algorithm. Not only is it able to handle larger ride sizes, but it can handle any size search significantly faster than Exhaustive. This is surprising that it is that much faster than Exhaustive

    b. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.

       Our Empirical analyses show an inconsistency between the actual Dynamic Algorithm's performance and it's expected performance given our mathematical analyses. This can be due to the caching method of the Dynamic Algorithm lowering the actual performance time.

    c. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.

       Hypothesis 1 states that "Exhaustive search algorithms are feasible to implement, and produce correct outputs." This is incorrect. Exhaustive Search has proven to be quite slower than Dynamic Programming.

    d. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.

       Hypothesis 2 states that "Algorithms with exponential running times are extremely slow, probably too slow to be of practical use." This is correct as it has taken much more time than Dynamic Programming.