Breadth First Search and STRIPS Operators: An Analysis of the Efficiency of Compound Goal States.

Benjamin Mark Hayward

School of Computing
Teesside University
P4117178@tees.ac.uk

Keywords: Breadth First Search and STRIPS Operators: An Analysis of the Efficiency of Compound Goal States. Module Code: VIS3071-N.

Abstract

This paper sets out to test the efficiency of the Breadth-first First (BFS) algorithm by applying a Clojure implementation of a BFS to STRIPs style operators. The problem space for this is pseudo-randomly generated from a NetLogo model of a warehouse, where automated forklifts are given various goals to accomplish. This is timed within the JVM, so as to ensuring that comparisons can be drawn between the efficiency of the BFS when working planning a route for a single forklift, and when planning routes for multiple objects. The goal of this study is not to measure effectiveness of the BFS, but to contrast the time taken for singular and compound goal-states to be reached. This study tests the hypothesis that compound goal states have an exponential effect on the time taken for a BFS algorithms operation.

1. Background

This study's focus is the comparison of the time taken for the BFS algorithm when applied to STRIPS-style operators; contrasting both singular and compound goal-states. This is demonstrated using a NetLogo (NL) model of a warehouse simulation, wherein forklifts are tasked in various ways by commands interpreted by a natural language processor (NLP).

1.1. Breadth-first Search

One advantage of a breadth-first search (BFS) graph traversal algorithm, is its inherent heuristic admissibility; ensuring that the shortest possible path is returned regardless of the scale of the problem-space. Applying a BFS to a large problem space however can prove costly, as the worst case time complexity means that the search can take up to as long as the sum of the graph's vertices and edges, or $T=O(V+E)$ *(Cormen and Leiserson, 2009).* As a width based algorithm, a BFS is generally more efficient than some other heuristic search algorithms, such as depth first search (DFS) when the goal node is close to start. In this study's context, this implies that the more STRIPS operators that have to be applied in succession to reach a goal state, the more nodes that have to be visited to reach the goal state. This raises the hypothesis that compound goal states have an exponential effect on the time taken for a BFS algorithm's operation.

1.2. Abstractions

So as not to add unnecessary complexities to the artifact; several concepts have been abstracted. The world knowledge is generated from the NL domain, composed into strings which are processed by Clojure, to return the path to the user, if one exists. One such abstraction is that within the Clojure knowledge-base, space is treated as non-linear, and operates in such a way that forklifts continually remain at various bays.

The ‘move’ operator offers a transition between bays, for example, from a loading bay to a shelf. This abstraction has been made to ensure that the problem is solvable within a reasonable timeframe, as a BFS would otherwise be traversing a exceptionally deep tree, making it incredibly time inefficient to use the BFS for spatial pathfinding; that kind of task would require a better suited search algorithm, like an A* search. The other operators ‘pickup’ and ‘drop’ take the same period of time as the the ‘move’ operator as far as Clojure is concerned, though in the NetLogo model, forklifts move in real-time. As the focal point of this study concerns the BFS, which is ran by Clojure on the JVM, this choice is purely aesthetics-based.

2. NetLogo front-end

A NetLogo (NL) model has been constructed for the front-end for this implementation, adding a GUI to the artifact, to give the STRIPS operators context; allowing for the route planning of autonomous forklifts in a warehouse environment.

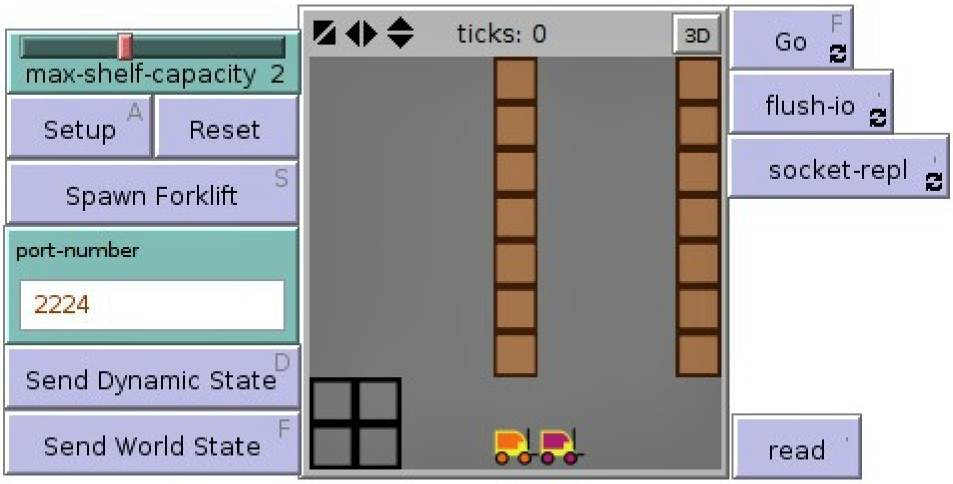

Figure 1– The NetLogo Interface.

2.1. NetLogo model

The NL world shows a top-down view of a warehouse floor, with the bottom left corner representing the loading bay of the warehouse, forklifts are represented as conventional manually operated forklifts, solely for aesthetic reasons and shelves and bays run in columns down the world. The purpose NL world is to run commands, that can be received as input, through communication with the artifact’s back-end. Such commands are used to maneuver the forklift around the world whilst interact with collectables.

2.2. Knowledge Base & Scale

The size of the NL world is kept small, though the model has been programmed to be adaptable to different world sizes. This study uses a world size of 8 x 8. This ensures that testing resolves in a reasonable amount of time, as this study is concerned with the impact of compound goals when applying a BFS to STRIPS operators. From the model, a knowledge-base is generated in NL; composing strings that can be used Clojure, as a collection of facts. through a socket to the JVM, using a pre-existing package Sock2 (Lynch, 2017), for an example of the rules within said knowledge-base:

```
((isa forklift (forklift 1))
 (adjacent (bay 17) (forklift 1))
 (holds (bay 17) (nails-crate 32))
)
```

Figure 2 - An Example of Rules Held in the Knowledge-base.

3. Clojure Planning

The planning is carried out using Ops-Search (Lynch, 2017); a tool that uses a BFS to find the path to a singular or compound goal states, by applying STRIPS-style operators to the knowledge-base generated by NL. Ops-Search continues applying operators until either all goal states are satisfied, or the search has reached the end of its maximum depth, having traversed every possible series of nodes.

3.1. Ops-Search, STRIPS Operators and Rule Application

Ops-Search uses cgsx.tools.matcher (Lynch, 2017), a symbolic pattern matcher, to apply STRIPS operators to the knowledge-base repeatedly, until the solution is found. The operator's consist of preconditions, that when successfully matched with the knowledge-base, append the additions to the knowledge base, and remove the deletions. Following this, text is printed to the console, and commands are evaluated by clojure if the goal-states have been resolved.

```
pickup{
    :pre( (adjacent (bay ?b) (forklift ?f))
          (isa bay (bay ?b))
          (on (bay ?b) (shelf ?s))
          (holds (shelf ?s) (?object ?o))
          (holds (forklift ?f) none)
          )
    :add( (holds (forklift ?f) (?object ?o)))
    :del( (holds (shelf ?s) (?object ?o))
          (holds (forklift ?f) none)
          )
    :txt  (Forklift ?f picks up ?object ?o from shelf ?s)
    :cmd  (sock2.socket/socket-write s25 (str '(pickup-collectable
              forklift ?f collectable ?o))))
}
```

Figure 3 - An Example of the STRIPS-style Operators used, in Clojure.

The planning mechanism takes singular or multiple goal-states, and iterates through the application of operators until the goal states have been satisfied. In the context of this study, this enables the measurement of the time taken by singular and multiple forklifts.

```
(
(holds (forklift 1) (object 1))
(holds (forklift 2) (object 2))
)
```

Figure 4 - A Compound Goal-state as represented in Clojure.

4. Clojure parsing

The back-end, is capable of parsing basic user input such as “Move to loading-bay”, or “Grab a nails-crate”. This is not a standard NLP, as it is not fully implemented, however is capable of translating constrained user inputted English to NL commands. the NLP uses a combination of lexical and semantic processing to take the user input and transform it into a command to be ran by NL. The input is first checked against the lexicon, to check a word’s given word type, and providing the matcher finds a match, the precedent facts on the right hand side of the ‘:=>’ operator are applied, ensuring that sentences are structured in a specific syntactical format for further processing.

```
(((-> ?v verb?) (-> ?d det?) (-> ?n noun?)) :=>
   (mout (list (?v) (? d) (? n))))

(((-> ?v verb?) (-> ?num num-txt?) (-> ?n noun?)) :=>
   (mout (list (? v) (? num) (? n))))

(((-> ?v verb?) (-> ?n noun?))  :=>
   (mout (list (? v) 'the (? n)))))
```

Figure 5 - Lexical Processing

The new sentence is then looked up again in the lexicon, and switched with a pre-defined synonym. In effect, this would take the command “lift box”, and output for further processing ‘(pick-up the crate), which in turn, is then parsed into a string representing an NL function, and sent to NL for evaluation.

5. Testing & Results

5.1. Method

Testing was carried out by simply timing functions executed in Clojure. All tests were ran 20 times; to ensure a thorough test, that still completes within a reasonable timeframe; Outliers are included in the weight of averages; as they are still valid data in the context of this study; some goal states simply take longer to search for. All results have been rounded to three decimal places for ease of digestibility, but the raw data can also be found within section 9.0, Figures.

5.2. Results

5.2.1. Singular vs Two Forklifts Moving

The first test that was ran was of one forklift moving to a random bay. As space is nonlinear, there is no need to use a specific bay. It consisted of triggering a single forklifts move operator, and following that, triggering two forklifts move operators.

	One Forklift, One Goal	Two Forklift, Two Goals
Mean:	19.854 ms	50.451 ms
Max:	40.281 ms	69.430 ms
Min:	14.103 ms	41.064ms

Table 1 - Single/Double Move Forklift Result Summary.

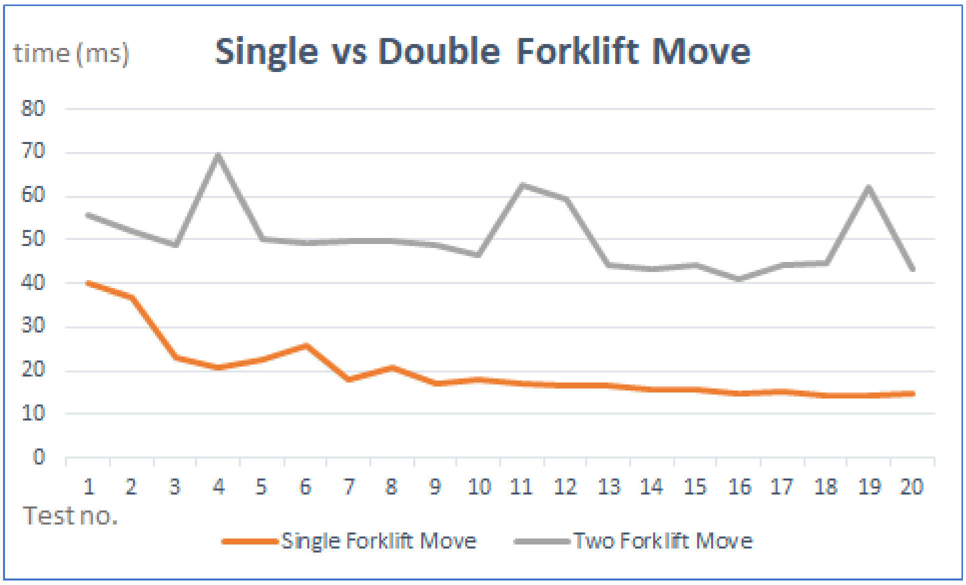

Figure 6: Single/Double Forklift Moving Comparison.

5.2.2. Single vs Dual Forklifts Moving and Picking-up an Object

To contrast further, a series of 20 tests were ran to wherein forklifts were given a goal of picking up an object. This requires the movement operator to trigger first, followed by the pickup operator.

	One Forklift, One Goal	Two Forklift, Two Goals
Mean:	145.981 ms	811.551 ms
Max:	113.372 ms	691.868 ms
Min:	94.442 ms	614.107 ms

Table 2 - Single/Double Move Forklift Result Summary.

These results indicate a 555.929% increase of the mean time taken for two forklifts over one, a 610.263% increase in the maximum time, and a 650.250% increase in minimum time.

5.2.1. Single vs Dual Forklifts Moving and Picking-up an Object.

The final test ran was the comparison of single and dual goal-states, when asking one or two forklifts to each put an item on the bay, and finally a test for each forklift to place two items on the loading bay, which consists of each forklift triggering two operators each.

	One Forklift, One Goal	Two Forklift, Two Goals	Two Goals Each:
Mean:	1015.413 ms	33357.946 ms	124374.884 ms
Max:	1274.516 ms	52411.419 ms	200405.845 ms
Min:	843.442 ms	15919.126 ms	57815.682 ms

Table 3 - Single/Double Forklift Move, Twice: Result Summary.

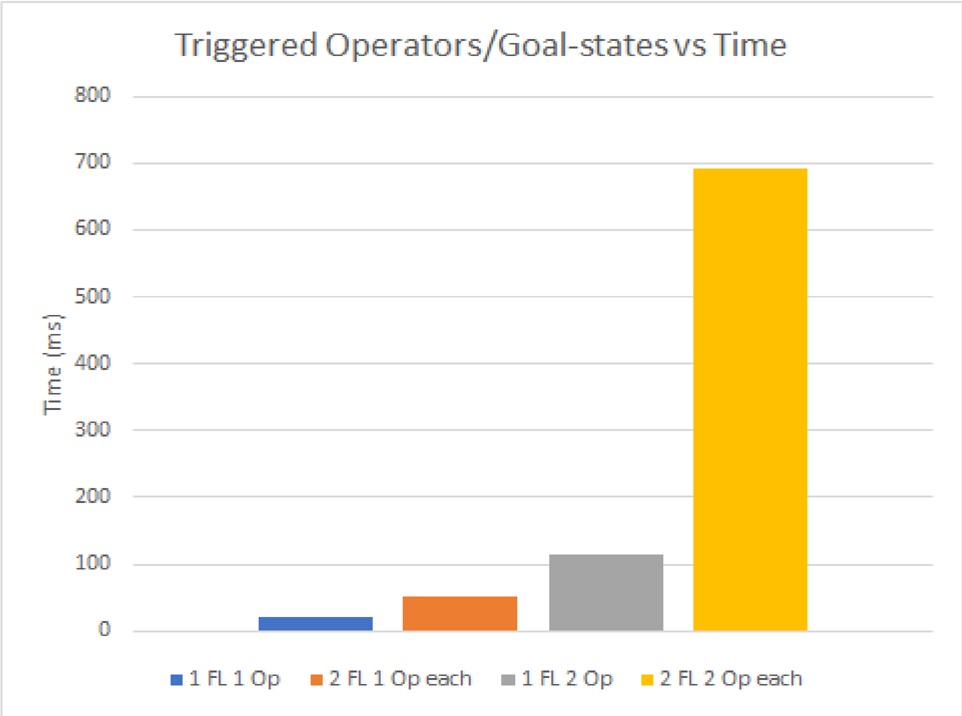

Figure 11.1: Triggered Operators/Goal-states vs Time.

5.3. - Miscellaneous Charts, Graphs and Tables

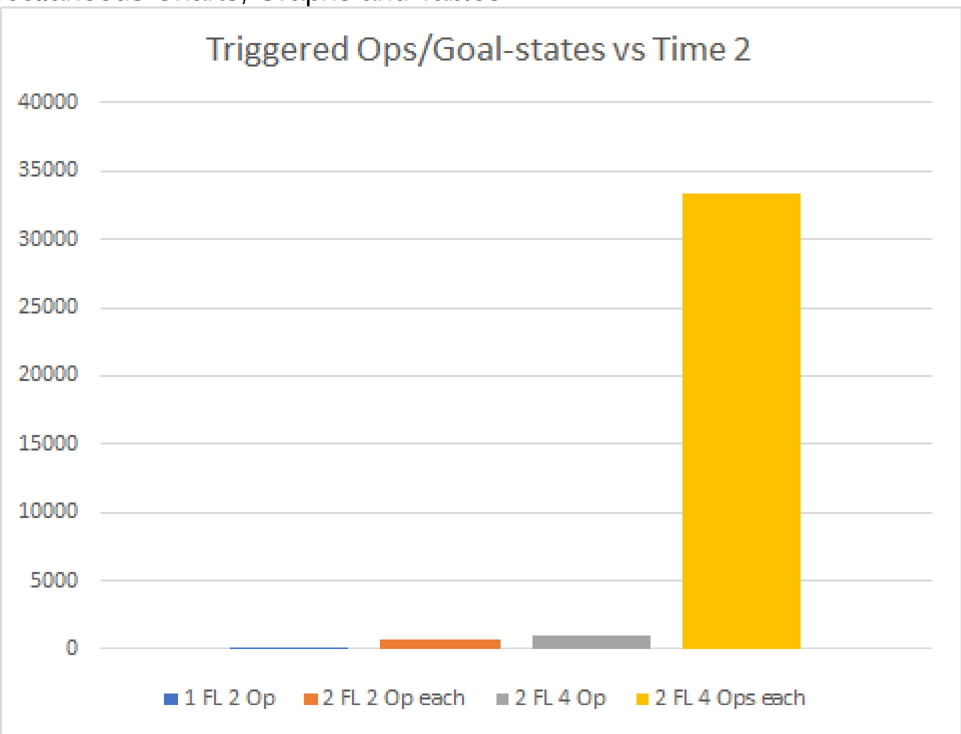

Figure 11.2: Triggered Operators/Goal-states vs Time - Part 2.

	Single Forklift , Two Goals	Two Forklifts, Two Goals	Two Forklifts, Two Goals Each:
Mean:	3285.160%	33357.946%	124374.884%
Max:	4112.261%	52411.419%	200405.845%
Min:	1887.400%	15919.126%	57815.682%

Table 4 - Percentile Increases of the Averages of All Tests.

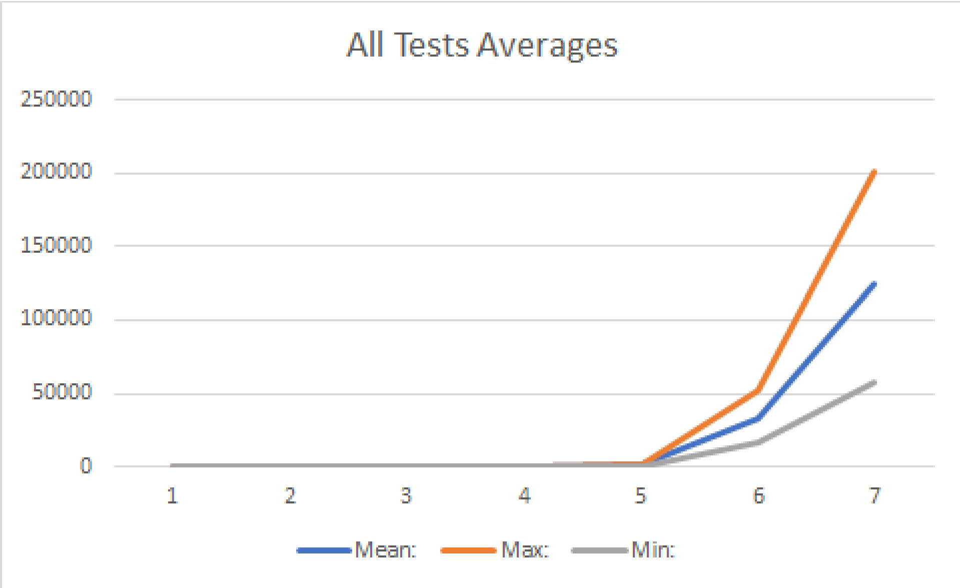

Figure 12: An Overview of all Timed Results

6.0. Analysis & Critique.

6.1. Analysis

The results in figure 6 and 7, indicate a an increase of 254.108%, in having multiple forklifts goal to move once. The maximum and minimum times increased by 172.365% and 291.172% respectively. Figure 7 confirms that there is a distinction between the singular and composite goal-states. In Fig 8 These results indicate a 555.929% increase of the mean time taken for two forklifts, a 610.263% increase in the maximum time, and a 650.250% increase in minimum time. This continues to show the link between the tree's depth and the goal state, supporting figure 6 and 7.

Figure 9 shows the relation of time for a single forklift moving to and picking-up an item, two forklifts each with their own goal state, and two forklift each with two goal-states. The mean increase in time for singular to double, double to quadruple, and singular to quadruple goal-states are 3285.160%, 4112.261% and 1887.400% respectively. (Figure 10). Figure 11 and 12 are included to show to exponential growth of the time of the problem space, only in the latter however is the final test included, as it eclipses the smaller values and makes the charts indigestible.

In the case of multiple forklifts, comparing the first test, the increase in time for adding a forklift with an equivalent goal is 254.110%. For the second, triggering two operators, a 280.718% increase, and finally, for the third, triggering four operators, ignoring the two forklift/four operators each test, compound goal states have the effect of increasing 745.907%.

6.2. Retrospective

Retrospectively,would have made significantly more sense to tie the NLP to the Ops-Search, by parsing the user input into a goal state, rather than a string to be evaluated within NetLogo. This did not ultimately take away from the test, but could have been timed at parsing. It would have also been nice to compare other Algorithms, such as the depth first search, and the nature of the DFS implies that it could deal with the lengthier searches in more time effective manners.

6.3. Conclusion

It is is clear that compound goal states do have an impact on the time of BFS, when used in this context, however it appears that the depth of a tree, and by extension how far away the goal nodes are, is the primary impact on performance time. One interesting result, was to see increases, over 250% from even using one extra forklift, showing that compound goals do have a direct impact on the BFS.

7.0. Acknowledgements

Thank you to Alan Blackwell, for creating the original basis for the template used in this document, and also thank you also to Simon Lynch, for the modification and distribution of the modified version of Blackwell's template. Thank you also to Eleonora Bilotta, Thomas Green and Paola Kathuria for their help in defining and testing the original template.

8.0. References

[1] Cormen, T. and Leiserson, C. (2009). *Introduction to algorithms*, *3rd edition*. 3rd ed. Cambridge, Massachusetts: The MIT Press, pp.594-602.

[2] Bundy, Alan. "Artificial Intelligence Techniques: a Comprehensive Catalogue." Artificial [3] Intelligence Techniques: a Comprehensive Catalogue, Springer, 1997, pp. 12–12.

[4] Lynch, S (2017) ops-search. (Version 1.0)[Source Code]. https://github.com/cognesence/ops-search.

[5] Hoang, H., Lee-Urban, S., & Muñoz-Avila, H. (2005, June). Hierarchical Plan Representations for Encoding Strategic Game AI. In AIIDE (pp. 63-68).

[6] Hendler, J. A., Tate, A., & Drummond, M. (1990). AI planning: Systems and techniques. AI magazine, 11(2), 61.

[7] Edelkamp, S. (2014, May). Planning with pattern databases. In *Sixth European Conference on Planning*.

[8] Jones, T. Breadth-First Search.

[9] B. (2010, May 18). Algorithms – Depth/Breadth First Search. Retrieved August 10, 2017, from https://computersciencesource.wordpress.com/2010/05/18/algorithms-depthbreadth-first-search/

[10] T. (n.d.). Artificial Intelligence Popular Search Algorithms. Retrieved August 10, 2017, from https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_popular_search_algorithms.htm

9.0. Figures

9.1. Single vs Dual Forklift Move Results Table

T#	Single Forklift Move	Two Forklift Move
1	40.280539	55.618258
2	36.767852	52.09612
3	22.933457	48.820881
4	20.852811	69.429594
5	22.472845	50.36285
6	25.622957	49.074225
7	17.778189	49.514989
8	20.629139	49.537062
9	17.184589	48.60908
10	17.792876	46.672843
11	17.148735	62.638428
12	16.38353	59.448194
13	16.41061	44.362064
14	15.846209	43.445284
15	15.864787	44.171808
16	14.895519	41.064233
17	15.152655	44.269041
18	14.430687	44.441222
19	14.103084	62.202524
20	14.530763	43.238316
Mean:	19.85409165	50.4508508
Max:	40.280539	69.429594
Min:	14.103084	41.064233

Figure 9.1. Single vs Dual Forklift Move Results Table

9.2. Single vs Dual Forklift Pickup Results Table

T#	Single FL Pickup	Dual FL Pickup
1	113.076122	676.326758
2	105.464049	691.563427
3	119.493158	673.016207
4	98.763294	700.612994
5	98.729381	670.300154
6	127.376838	721.882629
7	103.328546	805.948521
8	139.388722	614.107246
9	94.441674	723.449327
10	104.218626	616.816662
11	130.615629	700.89559
12	101.477699	616.209805
13	101.658697	727.850564
14	135.008968	742.471602
15	104.43951	644.100959
16	104.389422	748.96496
17	137.312438	651.117537
18	100.094074	646.622656
19	102.18282	811.550876
20	145.981069	653.538931
Mean:	113.3720368	691.8673703
Max	145.981069	811.550876
Min	94.441674	614.107246

Figure 9.2. Single vs Dual Forklift Pickup Table

9.3. Single vs Dual vs Dual, with Two Drop-offs each Results Table

T#	Single FL Drop-off	Dual FL Drop-off	Dual FL Two Drop-offs
1	843.442499	15919.12568	104087.8191
2	971.768275	17718.62787	113101.5087
3	861.960841	19050.5167	125399.7775
4	1056.404238	20982.93648	134042.8235
5	880.5791	22774.25852	145449.1297
6	1026.55299	24655.16917	153456.0208
7	890.831793	26875.00654	162967.1136
8	903.099958	28385.91718	173316.7812
9	1056.811617	29906.33504	198495.2215
10	933.807873	31719.11036	200405.8447
11	1101.123058	33904.69714	193528.5957
12	945.501419	35785.08341	57815.68248
13	950.026057	37534.73117	65162.85166
14	1162.621686	39736.50691	74967.46308
15	1002.813102	41956.47619	79372.50762
16	1274.515666	43973.22899	86590.24671
17	994.810017	45573.17618	97373.56432
18	1248.077273	48062.59909	101818.1938
19	1028.867776	50234.01663	108569.4261
20	1174.639344	52411.41946	111577.1085
Mean:	1015.412729	33357.94694	124374.884
Max	1274.515666	52411.41946	200405.8447
Min	843.442499	15919.12568	57815.68248

Figure 9.3. Single vs Dual vs Dual, with Two Drop-offs each Results Table