Determining Optimum Path in Synthesis of Organic Compounds using Branch and Bound Algorithm

Diastuti Utami 13514071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514071@std.stei.itb.ac.id

Abstract*— Synthesis of an organic compound from a certain compound offers a wide variety of possibility. Getting the optimum yield in a synthesis reaction has been a concern for many years in the field of organic chemistry. Sometimes, the reaction can be more complicated that it requires some time to determine the most efficient path. Nowadays, there is a number of software that could assist in determining the optimum path, such as software offering a database of reactions that could be used in path determining. In this paper, we will discuss how branch and bound algorithm could find optimum path in synthesis of organic compound(s).

Keywords—branch and bound; computational chemistry;branch and bound algorithm;

I. INTRODUCTION

Organic chemistry is a field of chemistry that studies compounds and substances made of a chain of carbon atoms. Reactions in organic chemistry often considered "unique" because it have distinct mechanism and almost always not yielding 100% products.

The thing with organic reactions is that it takes a lot of time. Therefore, catalysts are often involved in many reactions, but as we knew, the cost of catalyst often not cheap. Another thing is, the reaction is not always effective. Suppose we want to synthesize compound A from compound B, it will only yield 45% compound A, meaning that if we predict we can get, say, one mole of A, in reality we only get 0.45 moles of A.

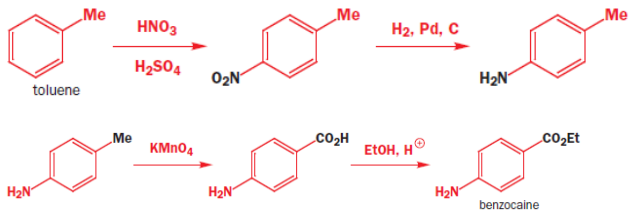

Figure 1 *Synthesis of benzocaine from toluene*

Organic compounds are often used in industrial scales, yet the process to synthesize an organic compound often involves many steps and takes a lot of time. On the other hand, there are many ways possible to synthesize an organic compound. This results in a challenge of getting the most efficient way to synthesize an organic compound.

Another challenge is to open all the possibilities of reactions. This is essential to determine the optimum path, a synthesis reaction often has many possibilities, but to create a comprehensive database of organic reactions is another challenge. Furthermore, we have to match the compound we want to search with the database—that's another thing.

The breakthrough in computational chemistry allows an insight to solve the challenge. Nowadays, many software offer assistant on searching the possible reactions, as database of organic reactions have been developed everywhere. Out of many possibilities to determine the most efficient way to synthesize an organic compound, we will see how the branch and bound algorithm solves the problem.

II. THEORY OF BRANCH AND BOUND ALGORITHM

Branch and bound is a state space search method in which all the children of a node are generated before expanding any of its children. It is similar to backtracking technique but uses BFS-like search. In branch and bound, we will use the term live-node to describe a node that has not been expanded, dead node which is a node that has been expanded, and solution node.

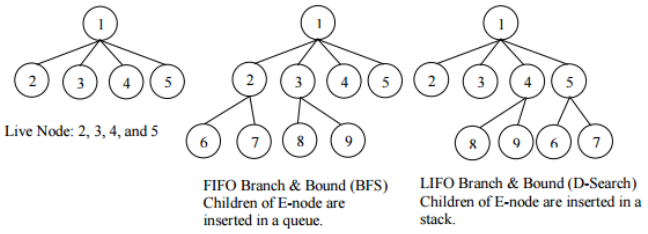

Figure 2 *FIFO and LIFO Branch & Bound*

The searching used in branch and bound is called least-cost search (LC). Least-cost search have some attributes:

- The selection rule for the next E-node in FIFO or LIFO branch-and-bound is sometimes "blind". i.e. the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.
- The search for an answer node can often be speeded by using an "intelligent" ranking function, also called an approximate cost function C
- Expanded-node (E-node): is the live node with best C value
- Requirements:
    - Branching: A set of solutions, which is represented by a node, can be partitioned into mutually exclusive sets. Each subset in the partition is represented by a child of the original node.
    - Lower bounding: An algorithm is available for calculating a lower bound on the cost of any solution in a given subset.

Example is 8-puzzle, where:

- Cost function: C = g(x) +h(x), where h(x) = the number of misplaced tiles and g(x) = the number of moves so far
- Assumption: move one tile in any direction cost 1

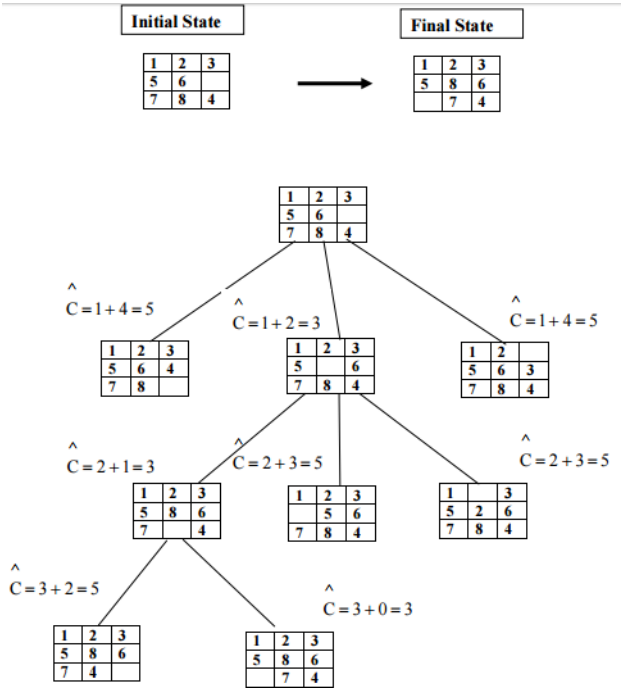

Figure 3 *8-puzzle branch and bound*

Global branch and bound algorithm:

1. Insert root node to queue Q. If root node is a solution node, then stop.
2. If Q empty and no solution found, then stop.
3. If Q not empty, select from Q a node i that have minimum cost C(i). if there is more than 1 node that satisfies the condition, choose one randomly.
4. If node i is a solution node, then stop. Else, generate all the child nodes. If there is no child found, return to step 2.
5. For each child j from node i, count C(j), and insert the children to Q.
6. Return to step 2.

The algorithm in this paper is similar to the knapsack problem. Knapsack problem is a problem where:

- Input
    - Capacity K
    - n items with weights w_i and values v_i
- Goal
    - Output a set of items S such that
        - the sum of weights of items in S is at most K
        - and the sum of values of items in S is maximized

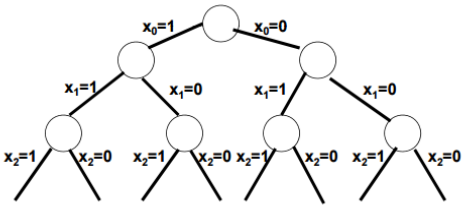

***Figure** 4 Tree for a knapsack problem*

Properties of a knapsack tree as follows:

- Left child is always xi = 1 and right child is always xi = 0
- Bounding function to prune tree:
    - At a live node in the tree, if we can estimate the upper bound (best case) profit at that node, and if that upper bound is less than the profit of an actual solution found already, then we don't need to explore that node.
    - We can use the greedy knapsack as our bound function as it gives an upper bound, since the last item in the knapsack is usually fractional
    - Greedy algorithms are often good ways to compute upper (optimistic) bounds on problems, e.g., for job scheduling with varying job times, we can cut each job into equal length parts and use the greedy job scheduler to get an upper bound
    - Linear programs that treat the 0-1 variables as continuous between 0 and 1 are often another good choice

Suppose we have a knapsack problem like in the table above, where maximum weight is 110. This generate a solution tree:

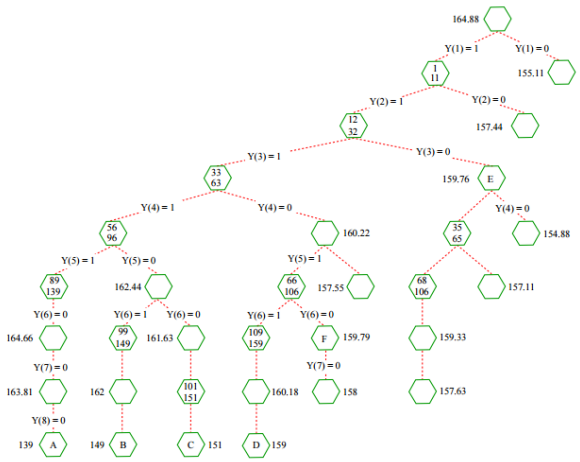

***Figure 5** Generated tree from the knapsack problem*

Where:

- Numbers inside a node are profit and weight at that node, based on decisions from root to that node
- Nodes without numbers inside have same values as their parent
- Numbers outside the node are upper bound calculated by greedy algorithm
    - Upper bound for every feasible left child (xi =1) is same as its parent's bound
    - Chain of left children in tree is same as greedy solution at that point in the tree
    - We only recompute the upper bound when we can't move to a feasible left child
- Final profit and final weight (lower bound) are updated at each leaf node reached by algorithm
    - Solution improves at each leaf node reached
    - No further leaf nodes reached after D because lower bound (optimal value) is sufficient to prune all other tree branches before leaf is reached
- By using floor of upper bound at nodes E and F, we avoid generating the tree below either node
    - Since optimal solution must be integer, we can truncate upper bounds
    - By truncating bounds at E and F to 159, we avoid exploring E and F

Branch and bound algorithm can also use depth first search. Depth first search is used in combination with breadth first search in many problems. Common strategy is to use depth first search on nodes that have not been pruned. This gets to a leaf node, and a feasible solution, which is a lower bound that can be used to prune the tree in conjunction with the greedy upper bounds. If greedy upper bound is less than lower bound, prune the tree. Once a node has been pruned, breadth first search is used to move to a different part of the tree. Depth first search bounds tend to be very quick to compute if we move down the tree sequentially, e.g. the greedy bound doesn't need to be recomputed. Linear program as bounds are often quick too: few simplex pivots.

III. BRANCH AND BOUND IN SYNTHESIS REACTIONS

First off, we should determine how we will count the bound. The factors related to the bound include yield, time, and cost. Because here we can't predict the cost, the bound will be determined using yield and time only.

Let's take a look at this simple reaction diagram. We can make a table out of it, consisting reagents, products, yield, time, and yield/time ratio. In this table "C" stands for compound, so C_1 means compound number 1, and so on. We will assume that the alkyl R is CH_3.

In real life, the diagram was generated through a searching in the database. A user will input a final product and some possible reagent that he/she has, and then from the database, we will get a few possibilities available from reagents to produce a certain product. The reaction path is represented with a directed graph.

Reagent	Product	Yield	Time(m)	Yield/Time
C_2	C_1	90%	60	1.500
C_2	C_3	82%	35	2.343
C_2	C_4	78%	75	1.040
C_2	C_5	75%	120	0.625
C_2	C_7	85%	80	1.063
C_3	C_6	62%	95	0.653
C_3	C_7	46%	120	0.383
C_5	C_2	70%	120	0.583
C_7	C_6	73%	34	3.042
C_7	C_8	80%	40	2
C_7	C_9	79%	60	1.317
C_7	C_{10}	60%	55	1.091
C_7	C_{11}	55%	25	2.200
C_9	C_{12}	74%	20	3.700
C_9	C_{13}	92%	30	3.067

Table 1 *Yield and time data table of reaction map in figure 6*

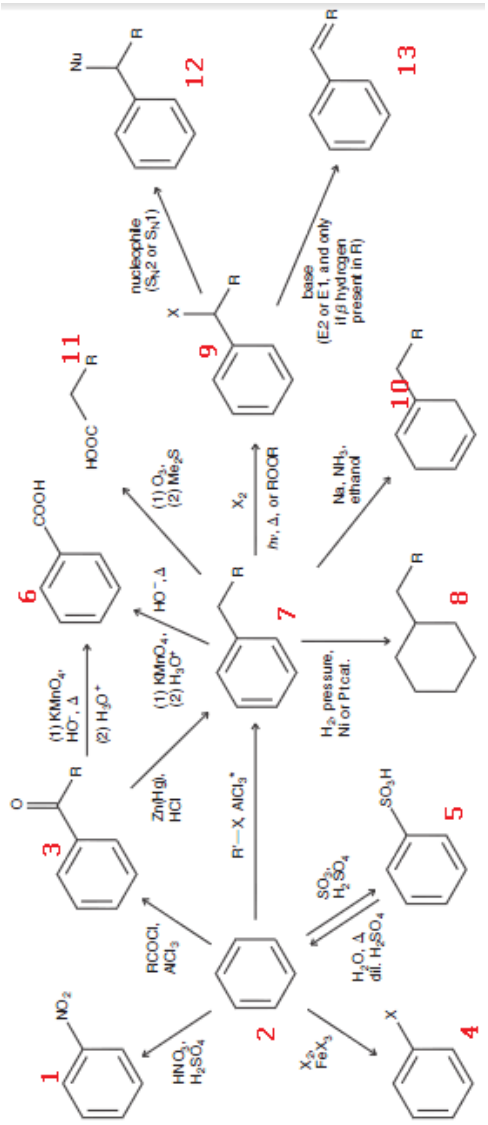

Figure 6 *Reaction Map of Aromatic Compounds*

As this problem similar to the knapsack problem, we can derived the bound formula as

bound = total yield + ((∑time – time taken) * next best yield/time)

with

total yield = current yield * selected reaction yield

There are a few differences from the knapsack problem, such as:

- The next best yield/time can only be determined by accessible compound from the reagent compound.
- The final bound calculation use the yield/time for final node to replace the next best yield/time

- We assume that the nodes accessible from a certain node have been determined when we retrieved the data from the given database.
- If there was no further reaction possible from a node, and the node isn't solution node, then deactivate the node.

Suppose we want to synthesize compound 13, and we don't have any compound 7 yet (and later compounds), there are few possibilities. First, we can start from compound 2, 3, or 5. The process of getting the result will be summarized on the table below.

Expanded Nodes	Active Nodes
C_0	C_2 = 0 + (969 * 2.343) = 2270.367 C_5 = 0 + (969 * 0.583) = 564.927 C_3 = 0 + (969 * 0.653) = 632.757
C_3, C_2	C_{36} = 0.62 + ((969 – 95) * 0) (dead because no further reaction possible) C_{37} = 0.46 + ((969 -120) * 3.042) = 2583.118 - C_{21} = 0.9 + ((969 – 60) * 0) = 0.9 (dead because no further reaction possible) C_{24} = 0.78 + ((969 – 75) * 0) = 0.78 (dead because no further reaction possible) C_{27} = 0.85 + ((969 – 80) * 3.042) = 2705.188 (C_5 dead because reaction possible is only to compound 2 and compound 2 already accessible from the start)
C_{27}	C_{37} = 2583.118 C_{276} = 0.85 * 0.73 + ((969 – 80 – 34) * 0) = 0.621 (dead because no further reaction possible) C_{278} = 0.68 (dead because no further reaction possible) C_{279} = 3067.9715 C_{2710} = 0.51 (dead because no further reaction possible) C_{2711} = 0.468 (dead because no further reaction possible)
C_{279}	C_{37} = 2583.118 C_{27912} = 0.497 (dead because no further reaction possible) C_{27913} Solution found, bound = 2451.151
C_{37}	C_{27913} = 2451.151 C_{378} = 0.368 (dead because no further reaction possible) C_{379} = 2919.6634 C_{3710} = 0.276 (dead because no further reaction possible) C_{3711} = 0.253 (dead because no further reaction possible)
C_{379}	C_{27913} = 2451.151 C_{37913} = 2328.187
C_{27913}	Final solution found. Node C_{37913} dead because the bound is lower. No other node active.

Table 2 *Branch and Bound Decomposition of the Aromatic Reaction*

From the table above, we can determine the solution is C_2 – C_7 – C_9 – C_{13}, which has the optimum path considering the yield and time taken.

We should note that the result on the table above is generated by only considering yield (number of product that can be generated) and time taken. In reality, many things should be taken into account and there should be a way to make a quantification of those things, such as cost, accessibility to the product, and so on. By looking at the diagram, we could see that this result is valid because the only possible way to synthesize compound 13 is by is C_2 – C_7 – C_9 – C_{13} or is C_3 – C_7 – C_9 – C_{13}, and it is clear that the yield and time taken in via C_2 is much more effective than via C_3.

In real life, the algorithm itself can be applied to a more complicated reaction map, such as:

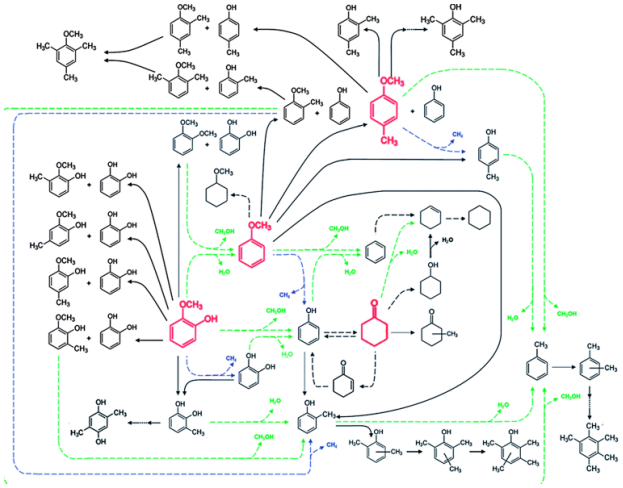

Figure 7 *A More Complicated Reaction Paths*

which won't be discussed here because there are numerous possibilities and it will be too long.

IV. CONCLUSION

By using branch and bound algorithm strategy, the bound can be counted by using the formula:

bound = total yield + ((∑time – time taken) * next best yield/time)

with

total yield = current yield * selected reaction yield

and is proven valid as it could generate the optimum path of the reaction on figure 6.

The branch and bound algorithm can find the optimum path in organic reactions, given many possibilities, yield, and time taken. This algorithm can be improved by considering the cost or money to undergo a reaction. This is also essential because in industrial world, we should make a production not only time effective, but also cost effective.

V. ACKNOWLEDGEMENT

I thank Allah SWT for his will so I can finish this paper and my parents who had always supported me. I also want to send my gratitude to Mr. Rinaldi Munir and Mrs. Nur Ulfa Maulidevi as IF 2211 lecturers for both have taught Algorithm Strategy.

I would also like to thank all the people who were involved in the process of writing this paper. Thank you for the people who have helped me both directly and indirectly. I hope this paper I wrote will help people who read it to gain more knowledge like I did while writing it.

REFERENCES

[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Teknik Informatika Institut Teknologi Bandung, 2009.

[2] Solomon, Graham T.W. Organic Chemistry. John Wiley & Sons, 2011.

[3] J, Clayden, et al. Organic Chemistry. Ocford, 2001

[4] Zumdahl, Steven S, Susan A. Zumdahl. General Chemistry. Brooks & Cole, 2010

[5] http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-204-computer-algorithms-in-systems-engineering-spring-2010/lecture-notes/MIT1_204S10_lec16.pdf

[6] https://www.seas.gwu.edu/~bell/csci212/Branch_and_Bound.pdf

[7] http://stanford.edu/class/ee364b/lectures/bb_slides.pdf

STATEMENT

I hereby declare that this paper is my own work and not a copy, translation, nor plagiarism of somebody else's work.

Bandung, May 9 2016

Diastuti Utami 13514071