A TWO-STEP ALGORITHM FOR A COMPLEX PROBLEM OF SHORTEST PATHS ON WEIGHTED GRAPHS AND 0-1 KNAPSACK

NADAV VOLOCH

Dept. of Computer Science, Sapir Academic College, Israel.
E-mail: nvolloch@yahoo.com

Abstract- two problems that are often studied and researches in computer science algorithms are the knapsack problem and the shortest paths on weighted graphs problem. Their combination is often addressed by dynamic programming solutions for the knapsack problem, using shortest path problem, with the creation of a knapsack graph. But these researches consider only two aspects of weight and value for an item/vertex, and here we address a different kind of problem in which we are taking into consideration three properties: item weight, item value and edge weight (that connects two items, but its weight is not depended on its vertices). The complex problem that combines these two, as a two-step problem on the same entities and a two-step algorithm for solving it, are the main ideas and subjects of this paper, in which this combination will be presented, along with several examples for it.

Index terms- Knapsack problem, Shortest paths on weighted graphs, Dijkstra's algorithm, 0-1 knapsack problem, graph theory, dynamic programmingy.

I. INTRODUCTION

The research is within the scope of theoretical algorithms.
In this branch, there are two well known problems that are:

a. The knapsack problem dating back far as more than a century ago ([6]), which is that for a set of items, each with a weight and a value, we have to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
To this problem there are three main versions:

1. The most common problem being solved is the 0-1 knapsack problem, which restricts the number of copies of each kind of item to 0 or 1 (hence its name), meaning we are able to pick only one item from each kind. This is the specific problem to which this paper addresses.

2. The bounded knapsack problem (BKP) in which there is no limit of one copy per item, but a limit for the number of copies of each kind of item extends to a maximum finite amount.

3. The unbounded knapsack problem (UKP) places no upper bound on the number of copies of each kind of item.

b. The shortest path problem is the problem of finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized as summarized in [1].

There are several algorithms that handle these problems, some more efficient than others.
For example, to UKP there is a greedy approximation algorithm proposed by George Dantzig ([2]). His version sorts the items in decreasing order of value per unit of weight, v_i / w_i, where v represents the value of each item i of n items (for $1 \leq i \leq n$) and w represents its weight . It then proceeds to insert them into the sack, starting with as many copies as possible of the first kind of item until there is no longer space in the sack for more. Provided that there is an unlimited supply of each kind of item, if m is the maximum value of items that fit into the sack, then the greedy algorithm is guaranteed to achieve at least a value of m/2.

We focus on two algorithms for these problems that are:

a. A pseudo-polynomial dynamic programming algorithm, suggested in different papers such as [8], in which at every step, the maximization process is re-assessing the current status limit-wise, and optimizing the result, by picking the best item for the current stage.
b. Dijkstra's algorithm, conceived by Edsger W. Dijkstra ([3]) that is an algorithm for finding the shortest paths between nodes in a graph, which is optimized in many researches such as [7] .

The research combines the two problems and algorithms to a two-step problem that is described in the next part of this paper. There have been papers about dynamic programming solutions for the knapsack problem, using shortest path problem, with the creation of a knapsack graph such as [4] and [5], but these papers gave a solution to the knapsack problem using the shortest path problem, thus considering two aspects of weight and value, as opposed to the complex problem presented here, taking into consideration three properties: item weight, item value and edge weight (that connects two items, but its weight is not depended on its vertices).

II. THE COMPLEX PROBLEM

A. Description and notations-building the graph

Given a weighted graph G = (V, E), where V represents the vertices of the graph, and E its edges, we first constrain the edges' weights to be non-negative ones, for using Dijkstra's algorithm. The second stage is adding properties of weight (w_i) and value (v_i) to each vertex, turning it to a legitimate knapsack item. We take into consideration the difference between the weight of the vertex, that is marked w, as mentioned before, and the weight of the edge, for the shortest path problem (manifested in Dijkstra's algorithm), that will be marked as w^E. In addition to these properties, we will add to the graph G a limit m, that represents the knapsack limit of maximum weight. We now have a suitable graph for both problems (knapsack and shortest paths in weighted graphs), that will be marked G^D.

B. The complex two-step problem

Given a graph G^D as described above, we wish to find the maximum value of items (vertices), with a knapsack weight limit (m), from the shortest path (of w^E) between two chosen vertices (source vertex-v_s and target vertex-v_t).

The first step is finding the shortest path (minimum edge weight) between two chosen vertices as shown in and the second step is considering the resulted sub-graph (the path), as items (the vertices) for the knapsack problem. An example of such a G^D is shown in Fig.1, were vertices are presented as V(v_i/ w_i,).

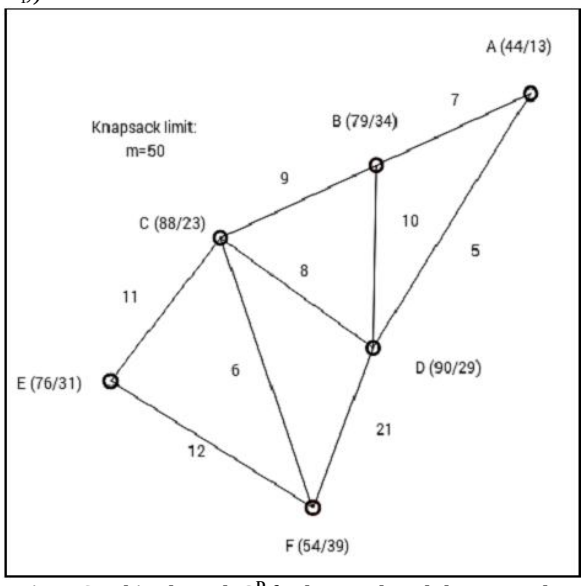


Fig. 1. Combined graph G^D for knapsack and shortest paths

III. THE ALGORITHM FOR THE COMPLEX PROBLEM

For solving the complex problem, we use a two-step algorithm: first we find the shortest path between two vertices in G^D, by using Dijkstra's algorithm, and then we take the vertices from the chosen sub-graph (the path), limiting the vertices weights with m- the knapsack limit, and then we use the pseudo-polynomial dynamic programming algorithm, to choose the vertex-items from the path.

The algorithm is as follows:

Complex knapsack-shortest path algorithm (Graph G^D, Vertex source, Vertex target, integer max_item_weight):

- Create sub-graph G^{path} = Dijkstra (G^D, source, target)
- Create chosen vertex-items list L^D = Knapsack 0-1 (V (G^{path}), max_item_weight)
- Return L^D

For example, given the graph G^D, shown in Fig.1, using A as the source vertex and F as the target vertex, the first step is finding the shortest path between them using Dijkstra's algorithm, we can see that the path is A-D-C-F, with a total edge-weight of 19. The chosen path is shown in Fig.2, thus we can see the graph G^D, after performing the first step in the complex algorithm.

Now, for the second step, we use the sub-graph G^{path} and take the vertex- items in it - V (G^{path}), for the knapsack problem, and considering the max weight limit m, as shown in Fig.3.

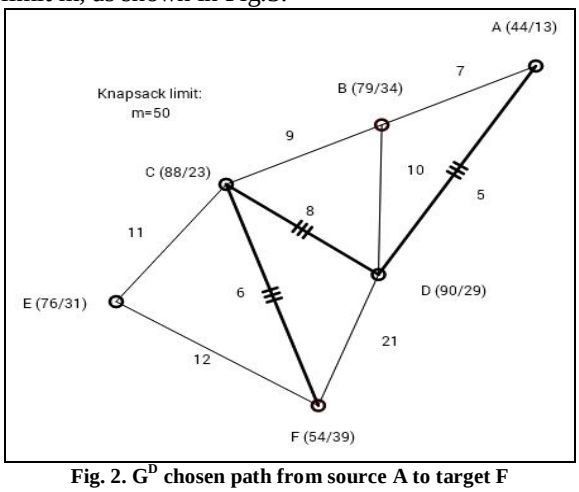


Fig. 2. G^D chosen path from source A to target F

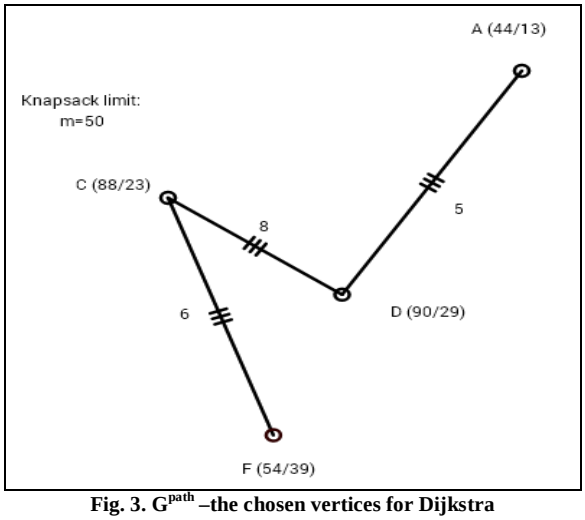


Fig. 3. G^{path} –the chosen vertices for Dijkstra

The vertex-items are: V (G^{path}) = {A (44/13), C (88/23), D (90/29), F (54/39)}.
For the pseudo-polynomial dynamic programming algorithm, we now re-assess the current status limit-wise at every step, thus manifesting the maximization process, and optimizing the result, by picking the best item for the current stage.

We do it by picking the items with most value (v), as long as we don't exceed the weight limit m (in this example m=50), thus choosing A and D, and their total weight is 42, meaning no other items that can be picked without exceeding m, achieving a total value of 134, as shown in Fig.4, with the chosen item-vertices shaded.
These are the items eventually picked from the shortest path of the source vertex to the target vertex, taking into consideration the maximal value of item-vertices of the path, thus providing a solution for the complex problem of knapsack and shortest path in a graph that has the three properties of item weight, item value and edge weight as explained above.

IV. MORE RESULTS AND EXAMPLES FOR G^D

In Table I, we can see a combined table for the complete two-step algorithm, in which the first part describes the item-vertices by their properties of value (v) and weight (w) and the edges (source vertex – target vertex) by their weight (w^E), then the source vertex and the target vertex are shown, for finding the shortest path between the (Dijksta), and the knapsack weight limit (m), and the last part describes the chosen item-vertices from the path, after using the pseudo-polynomial dynamic programming algorithm for solving the knapsack problem, and showing the total value of items.

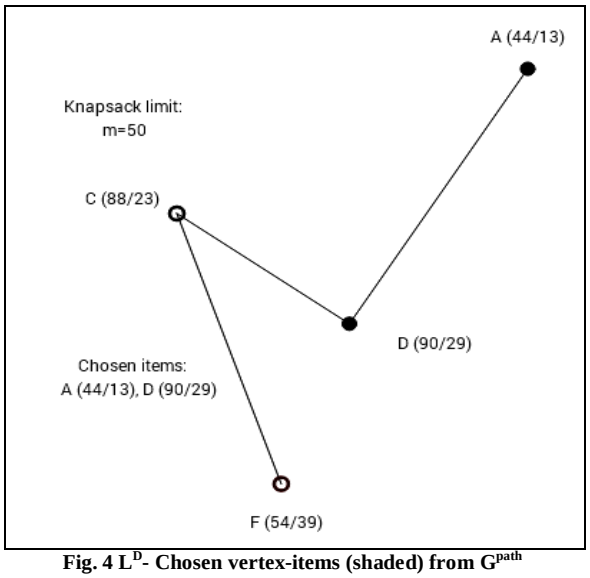


Fig. 4 L^D- Chosen vertex-items (shaded) from G^{path}

Table II is ordered in the same manner and displays a different, smaller graph G^D and its result after performing the complex two-step algorithm.

V. SCOPE AND OPTIMIZATION OF THE ALGORITHM

Since the algorithm creates a dependency of stages in which the shortest path problem is activated first, it may not achieve optimal results value-wise considering the knapsack problem. In that kind of solution the algorithm actually gives priority to a minimum edge-weight sum, over the maximal value sum for the knapsack items. For achieving optimal results in the complex problem of shortest path with maximal item value from a source vertex to a target vertex, a dynamic programming algorithm with both aspects should do the work. Since the knapsack limit is a constant (making the vertices weights a constraint property), and the edge weights are supposed to be minimized whereas the values of the items are supposed to be maximized, we can make an adjusted calculation for these properties. Thus, the algorithm presented here is relevant to these cases in which there is a known priority for the shortest path problem over the knapsack maximum value of items.

VI. ONGOING WORK

As mentioned in the previous part, the solution of the problem as presented here is not optimal for the cases in which there is no known priority for the shortest path problem. A dynamic programming algorithm with both aspects (giving the same priority for achieving minimal edge weight and maximal knapsack-item value) is currently being worked on. A system, written in Java, was built for manufacturing the results shown in this paper, based on the complex two-step algorithm described in part III. Improving it as described here is a work in progress.

TABLE I. G^D SCHEME EXAMPLE NO.1

G^D-The combined knapsack and shortest path graph		
Vertices		
$V(G^D)$	v	w
A	32	14
B	44	20
C	38	17
D	81	30
E	71	25
F	59	24
G	94	37
H	67	29
Edges		
$E(G^D)$	w^E	
A-B	15	
B-C	21	
C-D	13	
A-C	9	

B-D	14
D-E	12
E-F	23
F-G	19
F-H	20
G-H	18
B-H	17
C-H	22
C-F	25

m	**Source vertex**	**Target vertex**
45	*A*	*H*

G^{path}-The sub-graph of the chosen path	**Total Edge weight**
A-C-H	31

Chosen item-vertices	***Total value***
A, H	99

TABLE II. G^D SCHEME EXAMPLE NO.2

G^D-The combined knapsack and shortest path graph		
Vertices		
$V(G^D)$	***v***	***w***
A	70	31
B	88	43
C	91	45
D	95	50
Edges		
$E(G^D)$	***w^E***	
A-B	12	
B-C	16	
C-D	18	
A-C	19	
B-D	14	

m	**Source vertex**	**Target vertex**
85	*A*	*D*

G^{path}-The sub-graph of the chosen path	**Total Edge weight**
A-B-D	26

Chosen item-vertices	***Total value***
A, D	165

ACKNOWLEDGMENTS

The research is supported by The Open University and Sapir Academic College, Israel. I would like to thank both institutions for their support.

REFERENCES

[1] Abraham, Ittai; Fiat, Amos; Goldberg, Andrew V.; Werneck, Renato F. (2010) "Highway Dimension, Shortest Paths, and Provably Efficient Algorithms". ACM-SIAM Symposium on Discrete Algorithms, pages 782-793.

[2] Dantzig, George B. (1957). "Discrete-Variable Extremum Problems". Operations Research 5 (2): 266–288.doi:10.1287/opre.5.2.266.

[3] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs" . Numerische Mathematik 1: 269–271.doi:10.1007/BF01386390.

[4] Frieze, A. M. (1976). "Shortest path algorithms for knapsack type problems" . Mathematical Programming 11: 150. doi:10.1007/BF01580382

[5] Handler, Gabriel Y., Zang, Israel (1980), " A dual algorithm for the constrained shortest path problem", Networks- Wiley Subscription Services, Inc., A Wiley Company 10: 293-309, doi: 10.1002/net.3230100403

[6] Mathews, G. B. (1897). "On the partition of numbers" . Proceedings of the London Mathematical Society 28: 486–490.

[7] Mehlhorn, K., Sanders, P. (2008). "Algorithms and Data Structures: The Basic Toolbox" .199-200. Springer.

[8] Pisinger ,D. (1993). " A Minimal Algorithm for the 0-1 Knapsack Problem " . Operations Research September-October 1997 45(5):758 doi: 10.1287/opre.45.5.758

★★★