CASH FLOWS MANAGEMENT ALGORITHMS AND THEIR APPLICATIONS

Manana Chumburidze, Tsira Beradze

Akaki Tsereteli State University
e-mails: manana.chumburidze@atsu.edu.ge, tsiraberadze65@gmail.com
Georgia

Abstract: This article focuses on the development of innovative methods of cash flow management in the credit agencies. The generalized fractional knapsack problem to solve optimization problem of credit report selection in the planning stage is investigated. The tools applied in this development are studied, based on the binary tree and priority queuing approaches. Cash flows accounting software is constructed**.** For implementation the binary heaps algorithms and their applications have been used. The results are presented on both the fundamental and applied level.

Key words: generalized knapsack problem, credit agencies, binary heaps, priority queue.

1. INTRODUCTION

The purpose of the work is to demonstrate opportunity of accounting software for small businesses of Credit Agencies (CA) to identify cash outflows, cash inflows and the risk of payment delays to establish the possibility of default and improve. In this paper, we provide an overview of priority queueing approaches to manage queues of debts-clients in organizing, prioritizing and monitoring the progress of dynamic Debtors Database (DD) [1].

We recall of credit flows problem such as an instance of the generalized fractional knapsack problem [2]. Credit reporting agencies have financial portfolio to manage cash flows among different outflows opportunities. This paper discusses greedy algorithm of forward approaches [3] to build a solution incrementally, by step wise optimization according to some local criterion.

Binary tree to describe optimal subset of selection candidates with a best cash flows in the planning stage is constructed. The greedy rule can be expressed using priority queue of possible candidates (partial solution) to build the solution [4]. So, each iteration of the greedy algorithm removes the next best candidate from the priority queue and checks if the solution can add the candidate to build a larger solution. Priority queue of credit requires by the relevant value is used to select of debts-clients by the activities.

In particularly this algorithm consists and formed rooted tree where root store records of total value cash inflow related to full cash out flows from financial portfolio. The leaves are the optimal set selected candidates (subset) which store information about credit reports of debtors (enquiry credit and probability value) in the planning stage. They are viewed as a possible to create priority queue of credit reports to get optimal management of cash flows in the planning stage.

2. NOTATIONS AND DEFINITIVE CONSEPTS

In this section new technique of design paradigm algorithms to solve discrete optimization problems of cash flows is created [5]. Cash Flow Binary Tree (CFBT) has been constructed (Fig. 1). This problem can be formulated and solved as an generalized integer program used of a binary tree formulation. In particulary forward approaches of greedy algorithm method to construct multi stage binary tree to modeling selection of requered candidates and priority queueing approaches for mapping cash flows to optimize a portfolio are discussed.

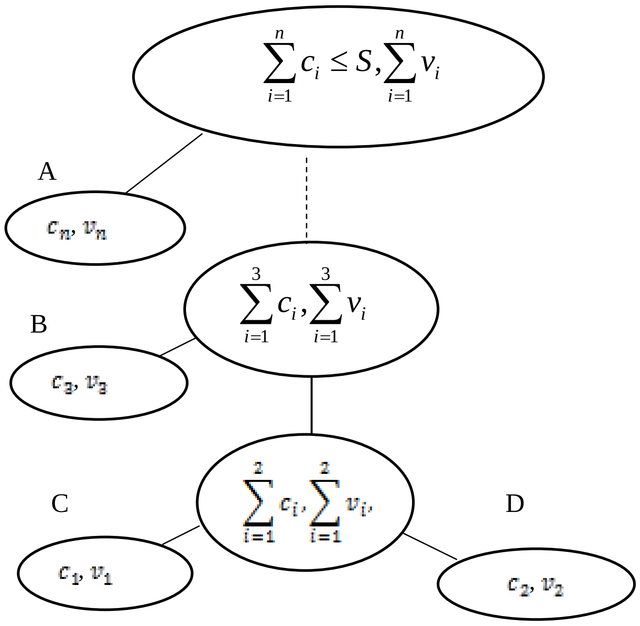

$\sum_{i=1}^{n} c_i \le S, \sum_{i=1}^{n} v_i$
A
c_n, v_n
$\sum_{i=1}^{3} c_i, \sum_{i=1}^{3} v_i$
B
c_3, v_3
$\sum_{i=1}^{2} c_i, \sum_{i=1}^{2} v_i,$
C
D
c_1, v_1
c_2, v_2

Fig. 1. Cash flow binary tree diagram

Let us introduce the following notations: S- capacity of financial portfolio. *n*-number of claims; $c_1, c_2 \ldots c_n$ -required credits; $v_1, v_2 \ldots v_n$ credit profits. An integer program for this problem would be:

$$\begin{aligned}&\text{Maximize}: v_1x_1 + v_2x_2 + .. + v_nx_n\\&\text{Subject to}: c_1x_1 + c_2x_2 + .. + c_nx_n \le S\\&x_j = \{0,1\},\ j = 1..n\end{aligned} \tag{1}$$

The technique works by creating a binary tree of nodes. These can be stored in a list, the size of which depends on the number of credit reports of debtors{\displaystyle n}. A node can be either a leaf node or an internal node. The process begins with the leaf nodes containing the records of enquiry credit and probability value of debtors in the planning stage.Then, the process takes the two nodes of enquiry credit (cash out flow) with smallest probability value (cash inflow)and creates a new internal node having these two nodes as children. The weight (cash in/out flows) of the new node is set to the sum of the weight of the childrens. Then the process on the new internal node and on the remaining nodes (i.e., we exclude the two leaf nodes) is applying again, this process repeats within constraint of financial portfolio. only one node remains, which is the root of CFBT and store total value cash inflow related to full cash out flows from financial portfolio in the planning stages.

Initially, all nodes are leaf nodes, which contain the **data** to store of value of required credit, the **weig**ht to store of related credit profits and the link to a **parent** node which makes it easy to calculate the length of leaf node.

The nods of CFBT defined as the follows:

```
struct node
{
      int  data;
      int   weight;
      node *left;
      node *right;
};
```

The outline of the algorithm solving selection problems of client rapports in order of priority queuing (PQ) approaches can be given as follows:

- Create a leaf node for each candidate and add it to the linked list
- Sort the list of records by increasing value-to-debt ratio
- Remove the two nodes of highest priority (lowest cashout flow) from the list
- Create a new internal node with these two nodes as children and with cash out flow equal to the sum of the two nodes' cash-out
- Add the new node to the list
- Sort current list
- Repeat compute the upper bound of the solution within kapasity of financial portfolio
- The least node of PQ is the root-node of binary tree.

The pseudo-code described as the follows:

```
bool  comp(node *a, node *b)
{
       return a->weight < b->weight;
```

```
};   //This function does a comparison in a way that puts smallest element before.

map <int, int> Map;
int main()
{
        list <node*> L;
// Create the link-list of nodes

        string s;
        cin >> s;

        for (int i=0; i<s.length(); i++)

                Map[s[i]]++;

for (map <int, int>::iterator it=Map.begin(); it!=Map.end(); it++)
                cout << (*it).first << " " << (*it).second << endl;
//  Create a map<> for each candidate

        for (map <int, int>::iterator it=Map.begin(); it!=Map.end(); it++)
        {
                node *root   = new node;
                root->data   = (*it).first;
                root->weight = (*it).second;
                root->left   = NULL;
                root->right  = NULL;
                L.push_back(root);

        }
 // Create a leaf node for each candidate and add it to the L;

            L.sort(comp);

        while(parent->data<=S)
        {if( L.size() != 1):
                {L.sort(comp);

                node *parent = new node;
                parent->left   = L.front();
                parent->weight = L.front()->weight;
               parent->data= L.front()->data;
                L.pop_front();
                parent->right   = L.front();
                parent->weight += L.front()->weight;
                parent->data += L.front()->data;
                L.pop_front();
                L.push_back(parent);

            }
// Remove the two nodes from the front of list and Create a new     internal parent
node with these two nodes as children and with data and weight equal to the sum of
the children nodes' dates and weights
};
```

- In the next stage and all the subsequent iterations compute the length of each leaf.

The pseudo-code described as the follows:

```
void counlength(node *parent, char c)     //count length of leaf;
{
        if (parent == NULL)
                return;

        if (parent->person == c)
        {
                length = count;
                return;
        }
        count++;
        countlength(parent->left,  c);
        countlength(parent->right, c);
        count--;
}
```

- Make Binary Heap (BH) implementation of PQ
  The BH data structure is used to modeling an efficient PQ for making the search space much smaller and reducing the calculation time.
  In this case the Min-Heap implementation is provided. All leaf nodes of CFBT will be included in PQ by length in increasing order and corresponding candidates will be served with the strategy FIFO (First-in, First-out) - see Fig. 2.

The pseudo-code described as the follows:

```
        Build-MinHeap (vector<int> leaf); //Find the minimal length's leaf-node
        Extract –Min (vector<int> leaf);  //Remove the minimal length's leaf-node
                                                                    from BH
        ENQUEUE (Q, leaf);  //Construct PQ with priority the minimal length's
                                                                   leaf-node
```

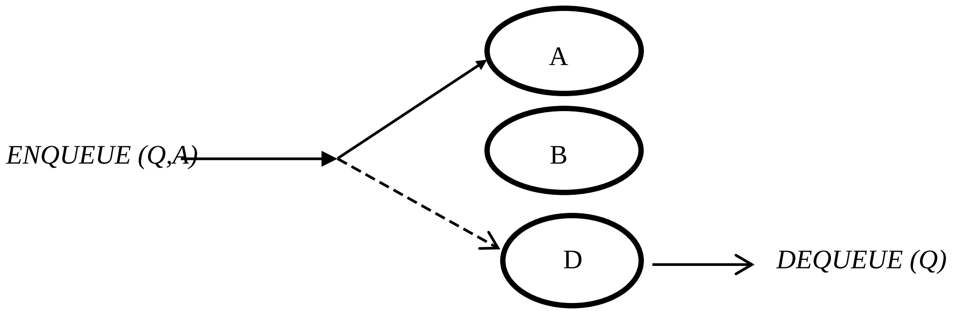


Fig. 2. Queuing Node

The time complexity analysis is as follows [6]:

- Extract-Min( ) is called **2 x (n-1)** times if there are **n** nodes.
- As Extract-Min( ) calls Min-Heapify( ), it takes **O(logn)** time.

Thus, Overall time complexity becomes O (n logn).

3. PROOF OF OPTIMALITY OF CASH FLOW BINARY TREE

This section concludes with a proof that the CFBT and PQ indeed gives the most efficient arrangements for the set of reports of debt clients.

Exactly, for any CFBT built by function containing at least two cash flow, the two record with least cash flow are stored in sibling nodes whose depth is at least as deep as any other leaf nodes in the tree.

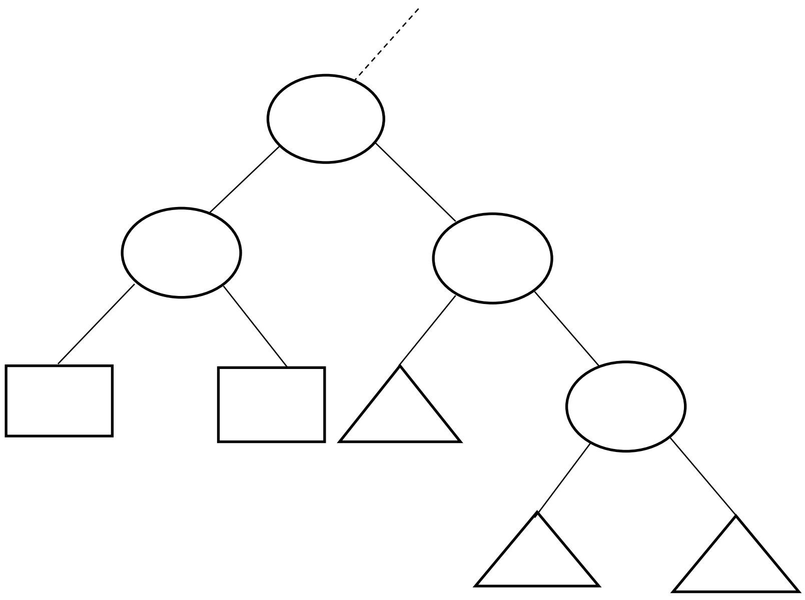


Fig. 3. An impossible CFBT

Proof: Call the two records with least cash flow c1 and c2. They must be siblings because CFBT algorithm selects them in the first step of the construction process. Assume that C_1 and C_2 are not the deepest nodes in the tree. In this case, the CFBT tree must either look as shown in Fig. 3, or effectively symmetrical to this. For this situation to occur, the parent of C_1 and C_2 labeled V, must have greater weight than the node labeled **X**. Otherwise, CFBT algorithm would have selected node **V** in place of node **X** as the child of node **U**. However, this is impossible because C_1 and C_2 are the records with least cash flow.

In Fig. 3 the triangles are representing of subtrees.

Accordingly to CFBT algorithm the leaf-node with minimal length is storing a best cash flow.

So PQ approaches optimize cash flow by save a leaf-node with lowest length in highest priority.

4. CASH FLOWS APPLICATION SOFTWARE

The best accounting software for small businesses needs to make the process as simple and easy as possible. This is as one of the most important and also challenging aspects of running a business is keeping good accounting records, especially being able to show your income, outcome, and profit at any one time.As a start-up it's easy to become overwhelmed with all the administrative tasks you need to manage. However, as your small business grows the volume of invoices can make it difficult to be entirely sure of the financial position of the business at any one time [7].

In this section cash flows manage software applications corresponding to the optimized algorithms represented in section 2 is demonstrated. Software package able to make monitoring cash outflows and inflows in CA and detailed analysis of cash flows, but helping to make better decisions by understanding cash flows. In this section cash flow management software by use the complex algorithms that are represented above is constructed.

The accounting software includes tools for analyzing data, making selections, forecasting and managing.

The software consists of three modules: new debits Module, cash inflow Module and cash out flow Module. which are equipped with the two section.

New debts registration window module the following sections (*Fig. 4.*):

- Section of dynamic database reports of new debts records containing information about the clients (personal data, location, telephone connections, requirement credit, credit history and e.t.c.)
- Section of new debts registration in current time including a buttons of new debts, delete, change data.

Cash outflows window involve the following subsection (Fig. 5.*)*:

- Section of dynamic database reports of credit requirements records
- Section for accounting of cash out flows, inflow's predictions and financial analysis in given time included the following buttons: cash outflow, delete,

change data and fields: back debts, total debts, percent debts, return percentage, date.

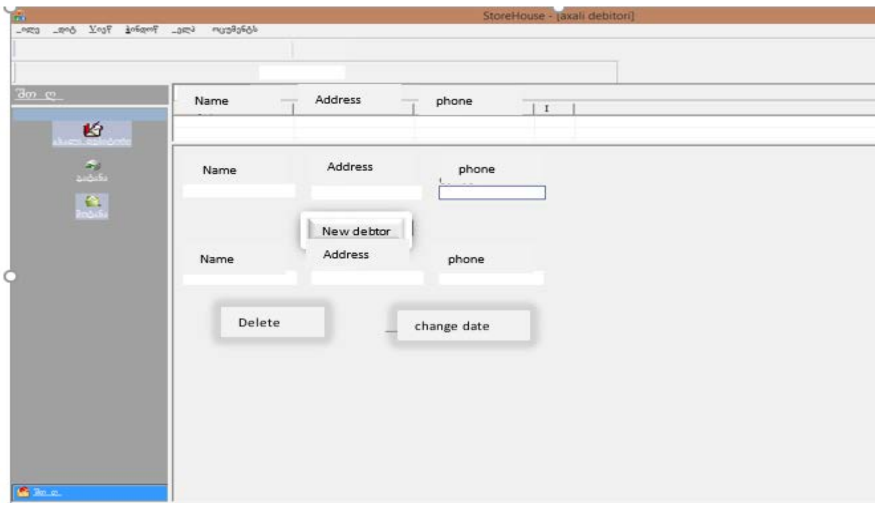
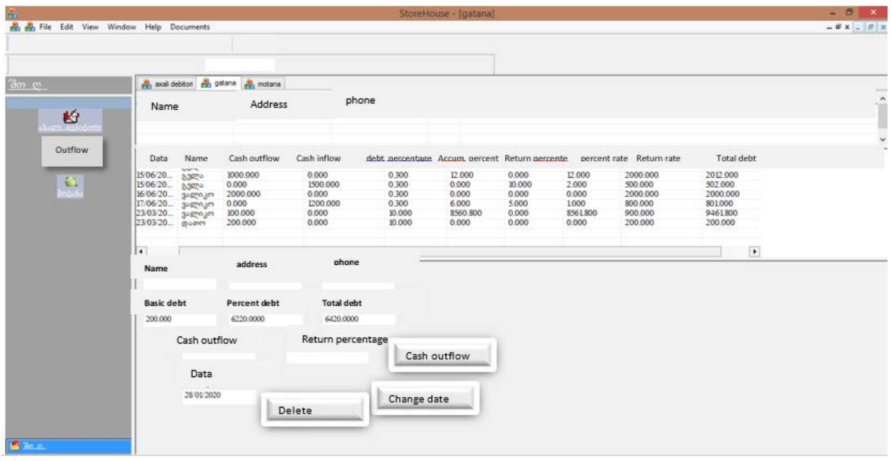


Fig. 4. New debts registration window



Data	Name	Cash outflow	Cash inflow	debt. percentage	Accum. percent	Return percente	percent rate	Return rate	Total debt
15/06/20...	[illegible]	1000.000	0.000	0.300	12.000	0.000	12.000	2000.000	2012.000
15/06/20...	[illegible]	0.000	1500.000	0.300	0.000	10.000	2.000	500.000	502.000
16/06/20...	[illegible]	2000.000	0.000	0.300	0.000	0.000	0.000	2000.000	2000.000
17/06/20...	[illegible]	0.000	1200.000	0.300	6.000	5.000	1.000	800.000	801.000
23/03/20...	[illegible]	100.000	0.000	10.000	8560.800	0.000	8561.800	900.000	9461.800
23/03/20...	[illegible]	200.000	0.000	10.000	0.000	0.000	0.000	200.000	200.000

Name	address	phone

Basic debt	Percent debt	Total debt
200.000	6220.0000	6420.0000

Cash outflow

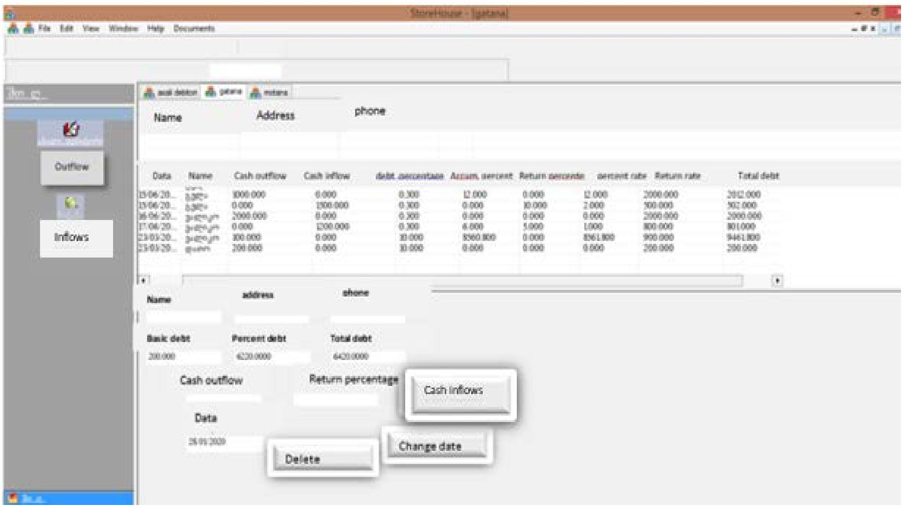
Return percentage

Data: 28/01/2020

Fig. 5. Cash outflows accounting window

Cash in-flows window involve the following subsection (Fig. 6.):

- Section of dynamic database of credit requirements reports.
- Section for accounting of cash inflows and trend analysis in current time included the following buttons: cash inflow, delete, change data and fields: back debts, total debts, percent debts, return percentage, date.



Name	Address	phone

Data	Name	Cash outflow	Cash inflow	debt percentage	Accum. percent	Return percente	percent rate	Return rate	Total debt
15/06/20...	[illegible]	1000.000	0.000	0.300	12.000	0.000	12.000	2000.000	2012.000
15/06/20...	[illegible]	0.000	1500.000	0.300	0.000	10.000	2.000	500.000	502.000
16/06/20...	[illegible]	2000.000	0.000	0.300	0.000	0.000	0.000	2000.000	2000.000
17/06/20...	[illegible]	0.000	1200.000	0.300	6.000	5.000	1.000	800.000	801.000
23/03/20...	[illegible]	300.000	0.000	10.000	8560.800	0.000	8561.800	900.000	9461.800
23/03/20...	[illegible]	200.000	0.000	10.000	0.000	0.000	0.000	200.000	200.000

Name	address	phone

Basic debt	Percent debt	Total debt
200.000	6220.0000	6420.0000

Cash outflow

Return percentage

Data: 25/01/2020

Fig. 6. *Cash in-flows accounting window*

4. CONCLUSION

Thus optimization problems of credit reports selection are investigated. Cash flows managing and forecasting algorithms in the planning stage are obtained. Priority queue scheduling approach for credit reports allocation in DD is created. For implementation BH algorithms and their applications are used.

Cash flow software package to collection of credit reports and compute Cash flows in CA by O (n log n) and O(n) computational complexity is constructed.

This implementation has a many advantages:

- Easy to understand and implement
- Identifies deliverables and milestones
- Forecasting and analyzing cash flow in any time
- Fluently make decision in the planning stage to select of candidates
- Monitoring and managing cash flows
- Optimize cash flow.
- Efficient in terms of time complexity

REFERENCES

[1] Goodrich, M.et al. *The Fractional Knapsack Problem*, *Algorithm Design: Foundations, Analysis, and Internet Examples*, John Wiley & Sons, 2002, pp. 259–260.

[2] Hans, K. et al. *Knapsack problem*. Springer Publishers; February 2004.

[3] Chumburidze, M. et al. Dynamic Programming and Greedy Algorithm Strategy for Solving Several Classes of Graph Optimization Problems. *Broad Research in Artificial Intelligence and Neuroscience.* **10** (vol. 1), Feb 2019, pp.101-107.

[4] Bennett, Nicholas. Introduction to Algorithms and Pseudocode, *Working paper in Project "Exploring Modelling and Computation"*, August 2015 (19 p.), DOI: 10.13140/RG.2.2.28657.28008

[5] Chumburidze, M. et al. About the Algorithms of Strategic Management. *The AMMCS 2019 Conference Book of Abstracts.* AMMCS 2019 Publishers, Waterloo, Canada, August 18–23, 2019, pp. 100-101.

[6] Chumburidze, M., I. Basheleishvili. The Complexity of Algorithms for Optimization Problems. *Computer Science & Telecommunications*, No.2(54). 2018, pp.125-130.

[7] Chumburidze, M. D. Lekveishvili. Numerical Approximation of Basic Boundary-Contact Problems, *ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Country, August 2017, pp. 1-7*;* DOI:10.1115/DETC2017-67097

Information about the authors:

Manana Chumburidze – PhD of Mathematics Sciences. Professor at the Akaki Tsereteli State University of Department of Computer Technology. Area of scientific research: Computer Sciences. Membership of The American Society of Mechanical Engineering (ASME). Member of the Scientific and Technical Committee of Editorial Review Board on computational sciences in the WASET organization of USA.

Tsira Beradze – PhD of engineering, invited teacher at the Department of Computer Technology of the faculty of Exact and Natural Sciences at Akaki Tsereteli State University.

Manuscript received on 10 February 2020
Revised version re-submitted on 07 March 2020