A Genetic Algorithm Solution to the Generalized ATM Cash Replenishment Optimization Problem

Kadircan Aksoy, Tarik Temur & I. Hakki Toroslu
Middle East Technical University Department of Computer Engineering
kadircan.aksoy@metu.edu.tr, tarik.temur@metu.edu.tr & toroslu@ceng.metu.edu.tr

Abstract

The ATM Cash Replenishment problem is a well known problem for the banking industry. Building on previous work by (Toroslu, et.al.), in this paper we propose a generalized version of the problem. Then we proceed by offering a novel Genetic Algorithm formulation of the problem and demonstrate its effectiveness.

Introduction

The ATM Cash Replenishment Optimization problem is a well known problem in the banking industry. Banks want to keep their ATM's loaded at all times, while minimizing the amount of idle cash that is in an ATM, since they can not make any interest money out of idle cash. However, loading itself also has a cost due to factors like transportation and security. This proposes an interesting game theoretical situation, where we have two conflicting costs that we want to optimize. We want to load an ATM as frequently as possible to minimize the interest loss while at the same time we want to load as seldom as possible to minimize the loading cost.

Problem Definition

Let I be a matrix of size DxD, where D is the length of the time interval which we want to optimize over, in days. The upper triangle of I, holds the values for the amount of money lost, for any subinterval of the original interval. So any entry $I[i, j]$, holds the amount of money lost if we were to load the ATM on day i, up to day j. The precise details on how this matrix I is calculated can be found in (Toroslu, et.al)

For the loading cost, we assume that loading any set of ATM's has a fixed cost, say α_n , where n is the number of ATM's being loaded together. This value is independent of the time interval in which we are loading the ATM's and does not linearly scale with n.

Methods

Dynamic Programming

We first look at the recurrence relation from (Toroslu, et.al) for the single ATM case, which can be implemented as a Dynamic Program:

$$c_1[i, j] = min \begin{cases} min_{i \leq k < j}(c[i, k] + c[k + 1, j]) \\ I[i, j] + \alpha_1 \end{cases}$$

Linear Programming

For the single ATM case, the problem can be expressed as the following Linear Program, where a decision variable x_{ij}, indicates whether the ATM has been loaded on day i, up to day j :

$$\sum_{i=1}^{D} \sum_{j=i}^{D} (x_{ij})(I[i, j] + \alpha_1)$$

subject to

$$x_{ij} \in \{0, 1\}$$

$$\sum_{k=1}^{D} x_{1k} = 1$$

$$\sum_{j=J}^{D} x_{Jj} - \sum_{j=1}^{J-1} x_{j(J-1)} = 0, \quad \forall_{1<J<D}$$

$$\sum_{k=1}^{D} x_{kD} = 1$$

Once a second ATM is introduced, the Linear Program has to be modified to handle different loading costs for loading the ATM's together or separately. The new decision variables are of the form $x_{ijk}^{(n)}$, where i and j denote the beginning and end of the loading period, $k \in \{0, 1\}$ denotes whether if both ATM's were loaded together or not and $n \in \{1, 2\}$ denotes the ATM number. The new objective function takes the following form:

$$\sum_{i_1=1}^{D} \sum_{j_1=i_1}^{D} x_{i_1 j_1 0}^{(1)}(I_1[i_1, j_1] + \alpha_1) + x_{i_1 j_1 1}^{(1)}(I_1[i_1, j_1] + \frac{\alpha_2}{2})$$
$$+ \sum_{i_2=1}^{D} \sum_{j_2=i_2}^{D} x_{i_2 j_2 0}^{(2)}(I_2[i_2, j_2] + \alpha_1) + x_{i_2 j_2 1}^{(2)}(I_2[i_2, j_2] + \frac{\alpha_2}{2})$$

Constraints are omitted.

Genetic Algorithm

Let a chromosome C be a DxN matrix of binary digits. Every row corresponds to a day and every column, an ATM.$C_{ij} = 1$ means that on day i, the ATM j has been loaded. For any column, (i.e any ATM) we can infer that an interval between two consecutive 1 entries, defines a load period. The number of 1 entries on a row directly gives us α_n, since it tells us the ATM's are loaded together on that day. From here, we can calculate the fitness of a chromosome with a simple $\mathcal{O}(DN)$ procedure. Below is an example chromosome with 4 ATM's and 6 days:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Results

For reasonable inputs, where $ND < \epsilon$ for a small ϵ, we were able to use an exhaustive approach to find the globally optimum loading schedule. In inputs of this size, we see that the Genetic Algorithm always finds the optimum solution in a few iterations, regardless of the interest rate or the loading costs.

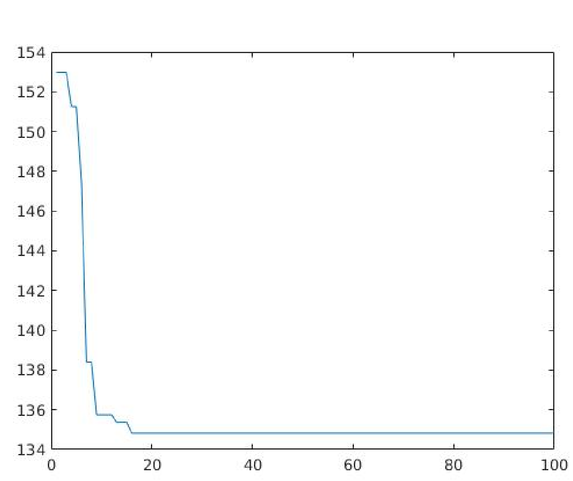

Figure 1: 5 ATM's 7 Days

For larger and more interesting experiments, unsurprisingly, as the solution space grows, the algorithm takes longer to converge:

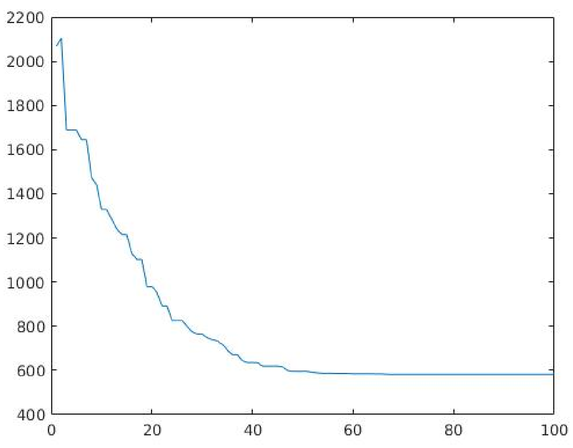

Figure 2: 15 ATM's 15 Days, % 1 interest

And also unsurprisingly, when the interest rate shrinks, the load costs become competitive with the interest loss and the algorithm has a harder time finding an optimum solution:

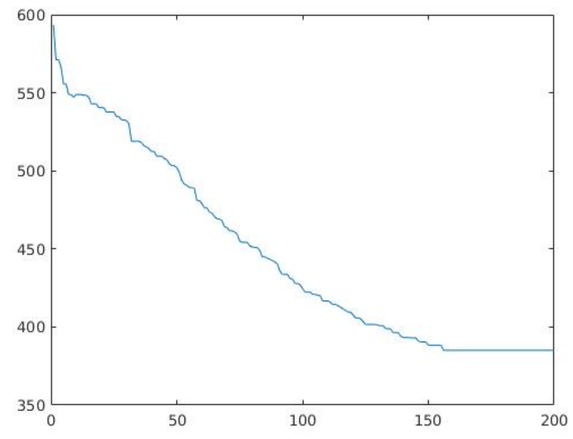

Figure 3: 15 ATM's 15 Days, % 0.1 interest

Conclusions

In our experiments, we have seen that in realistically sized inputs, the GA converges in at most a few minutes on an ordinary PC regardless of the problem setup. One can construct a general version of the LP from the one for 2 ATM's, however we believe that would be unnecessary since the GA is acceptably powerful. Further experimentation would most likely verify our results.

References

Ozer, F., Toroslu, I. H., Karagoz, P., & Yucel, F. (2018, October). Dynamic Programming Solution to ATM Cash Replenishment Optimization Problem. *In International Conference on Intelligent Computing & Optimization* (pp. 428-437). Springer, Cham.