Upon an Integer Allocation Problem

Claudiu VINTE
Goldman Sachs Ltd. Tokyo, Japan

Abstract: *A class of heuristic algorithms for trading execution allocations on investors' accounts.*
Keywords: *allocation, integer, alghoritms, stock exchange.*

Introduction

Trading activity in a brokerage firm implies, from the informational flow point of view, collecting the *orders* (for buying or/and selling various financial products) from firm's clients (designated, in the last instance, by their brokerage accounts opened at the brokerage firm - each client may have multiple brokerage accounts open with the brokerage firm) and placing these orders on a previously specified stock exchange. After the trade is made (the orders are executed, inside the stock exchange's matching engine), the *executions* are captured by the brokerage firm's trading system, and the executed quantities from each financial product (stocks, for instance) have to be allocated *fairly* on each client's accounts based on the ordered quantities for each account, specified previously by the client. Inside the trading system of the brokerage firm, the client's ordered quantities for each of it's accounts, are aggregated. Then trenches – or blocks – from the cumulated quantity are actually sent to the stock exchange. These portions, which have to be multiple of the *lot size* (the minimum number of the financial product units that is allowed to be traded on a particular stock exchange, and which is specified by the stock exchange's regulations for each product, based on it's price - l_i , $i=\overline{1,n}$) of the concerned financial product, are sent to the stock exchange through the brokerage firm's trading system, and they may be *fully executed*, *partially executed* or *not executed* at all. When these portions are executed, they may be executed at different prices, that is. Therefore, the total, aggregated quantity from a certain financial instrument, order by a certain client, may not be entirely executed (the client's orders not entirely satisfied) or executed in portions at different prices.

Formalizing the problem, we have (given) the requested (ordered) client's quantities, and for each product P_k, the actual executed quantities at their respective prices – the input data of the problem is described by the following bi-dimensional arrays. $C_1, C_2, \ldots, C_n$ ($n=\overline{1,n}$) are the client's accounts.

	Executed Quantity	**Price**
P_k	e_{k1}	p_{k1}
P_k	e_{k2}	p_{k2}
.	.	.
.	.	.
.	.	.
P_k	e_{kr}	p_{kr}

	C_1	**C_2**	. . .	**C_n**		
P_1	q_{11}	Q_{12}	. . .	q_{1n}	**l_1**	**Q_1**
P_2	q_{21}	Q_{22}	. . .	q_{2n}	**l_2**	**Q_2**
.	.	.		.	.	.
.	.	.		.	.	.
.	.	.		.	.	.
P_m	q_{m1}	q_{m2}	. . .	q_{mn}	**l_m**	**Q_m**

If the entire ordered quantity, from a certain product-client, it is executed (*fully* or *partially*) at unique price then, there is not an allocation problem. The executed quantity will be allocated on the client's accounts *proportionally* to the quantity ordered by the client for each of its accounts. If there is a price breakdown, i.e. the total ordered quantity from a certain product is executed (*fully* or *partially*) at multiple prices, then we have an integer allocation problem, which implies optimization. In this case, the final goal is to achieve average prices for each of the client's account as close as possible to each other, with the respect to the original ordered quantities for each account.

We have, therefore, the following constraints:

$$\sum_{h=1}^{r} e_{kh} \le Q_k = \sum_{j=1}^{n} q_{ij} \quad (k = \overline{1,m}) \cdot$$

Where ***r*** is the number of received executions for a certain product, ***m*** is the number of distinct financial products, and ***n*** is the number of the client's accounts. The quantities q_{ij}, e_{kh} $(i = \overline{1,m};\ j = \overline{1,n};\ k = \overline{1,m};\ h = \overline{1,r})$ are integers and they must be multiple of the corresponding product's *lot size* (l_i, $i = \overline{1,n}$, which are also integers).

Content

The proposed algorithms require two phases: the first one provides an initial basis solution, which will be improved, iteratively, in the second phase.

The client's accounts receive priorities function of the ordered quantity – a bigger ordered quantity implies a higher priority of satisfying the request, in the case of some *pro-rata* allocation approach, but for *small accounts priority* strategy, as the name itself suggests, a smaller ordered quantity implies a higher priority. As an additional note, each client's account may have assigned an explicit priority, which kicks in when two or several accounts have assigned the same ordered quantity for a particular financial product – in order to assure a rigorous manner of the request procesings.

	C_1	**C_2**	.	.	.	**C_n**
Account priority	x_1	x_2	.	.	.	x_n

The problem consists, in fact, in several local problems that may occur for each product-client pair, which may have a price breakdown. The algorithms are to solve each individual problem, and we shall focus hence on the allocation of executed quantities, at different prices, for a single financial product ordered by a certain client. Preparations - *the algorithms require some arrangements of the input data which serve for simplifying the processing:*

- sorting the accounts in an ascending order (descending order, for a certain *pro-rata* strategy), function of the ordered quantity and the explicitly assigned priority (where it is necessary);
- computing the matrix of the coefficients associated to the ordered quantities, as follows (requested by the pro-rata strategy basis allocation):

	C_1	**C_2**	.	.	.	**C_n**
P_1	s_{11}	s_{12}	.	.	.	s_{1n}
P_2	s_{21}	s_{22}	.	.	.	s_{2n}
.	.	.				.
.	.	.				.
.	.	.				.
P_m	s_{m1}	s_{m2}	.	.	.	s_{mn}

$$s_{ij} = \frac{q_{ij}}{\sum_{j=1}^{n} q_{ij}} = \frac{q_{ij}}{Q_i} \quad ; \quad i = \overline{1,m}$$

- sorting the executed prices per product in an ascending order, starting with the closest executed price to the general weighted average price, for each product, and going

toward the remoter executed prices from the average.

First phase – provides an initial solution – *rough allocation*. There are two strategies for obtaining an initial, basis solution.

Pro-rata strategy - it consists in the following steps and provides itself an acceptable solution in many concrete cases:

1. allocate the quantity executed at the closest price to the weighted average price, based on the coefficients determined previously for each product (they play the role of an optimality indicator, in the quest for the optimal solution), – *pro-rata* allocation, with the allocated quantity rounded to the nearest *lot size*;
2. there might be a difference between the quantity (the number of shares) executed at that price and the actual allocated quantity, caused by rounding errors; we call this difference *adjustment*, and we'll allocate the adjustment to the account with the biggest ordered quantity (with the respect to the original ordered quantity – to not be exceeded – and the explicitly assigned priority, where it is necessary); note that the *adjustment* may be distributed to several accounts, respecting the afore mentioned condition; after the adjustment is allocated the total number of ordered product's units should be equal to the number of units executed for that product, at that particular price;
3. for each account is calculated the *residual* value, as the number of product's units resulted by subtracting the number of already allocated units from the number of ordered units; the next price is allocated using this *residual* as criteria for updating the accounts' priorities;
4. repeat the previous steps to the remotest (from the average) executed price.

It has been proven empirically that this *pro-rata allocation* can provide itself an acceptable solution in many cases and, eventually, a very good initial basis solution for the second phase of the algorithm.

Small accounts priority strategy - it doesn't, necessarily provide itself an acceptable solution because the accounts with a bigger ordered quantity might be under satisfied, but the allocation has a great potential of being improved in the second phase of the algorithm; it consists in the following steps:

1. try to allocate the entire quantity executed at the closest price to the weighted average price, to the account (not yet satisfied) which has the smallest ordered quantity;
2. there might be a difference between the quantity (the number of shares) executed at that price and the actual ordered quantity fro that given account; we call this difference *adjustment*, and we'll allocate the adjustment to the next account (from the ascending sorted list of the accounts - ordered quantities - with the respect to the original ordered quantity – to not be exceeded – and the explicitly assigned priority, where it is necessary); note that the *adjustment* may be distributed to several accounts, respecting the afore mentioned condition, for each account we try to satisfy the entire ordered quantity at the closest possible price to the average;
3. repeat the previous steps to the remotest (from the average) executed price.

As we specified, this strategy for generating the initial basis solution, might not satisfy integrally the ordered quantity for the big accounts (especially when the executed quantity differs significantly from the originally ordered quantity), but this issue is resolved by the second phase of the algorithm, and this allocation turns out to be a very efficient initial basis solution for the optimization phase.

Second phase – concerns, in fact, the class of the heuristic algorithms that we have been intending to present. They follow, essentially, a *greedy* strategy. The algorithms provide an improvement of the solution at each iteration [1]. There is defined an objective function, which serves as criteria for continuing, respectively stopping, the iterative process. The

manner in which the objective function is defined, assures that with each iteration it is made a step forward toward a better allocation, although the optimality of the final solution is not necessarily assured [2, 3].

A. The objective function based on the total allocated quantity.

For ease, let's consider ***n*** the number of accounts, ***m*** the number of distinct prices at which the order for the product P_k was executed, and the allocated quantities a_{ij} ($i=\overline{1,m}$; $j=\overline{1,n}$). $P_k \rightarrow \begin{Bmatrix} E_1 & E_2 \dots E_m \\ p_1 & p_2 \quad p_m \end{Bmatrix}$, the pairs of executed quantities and the corresponding prices.

$A_j = \sum_{i=1}^{m} a_{ij}$ ($j=\overline{1,n}$).

	C1	**C2**	.	.	.	**Cn**	
E1	a_{11}	a_{12}	.	.	.	a_{1n}	p_1
E2	a_{21}	a_{22}	.	.	.	a_{2n}	p_2
.	.	.				.	.
.	.	.				.	.
.	.	.				.	.
Em	a_{m1}	a_{m2}	.	.	.	a_{mn}	p_m
	A_1	A_2	.	.	.	A_n	
	Δ_1	Δ_2	.	.	.	Δ_n	
	$\overline{p}_1$	$\overline{p}_2$	.	.	.	$\overline{p}_n$	$\overline{p}$
	V_1	V_2				V_n	

The steps of the algorithm are as followings:

1. for each product we have computed (at the time of the preparations), based on the ordered quantities, the coefficients $s_j = \frac{q_j}{\sum_{j=1}^{n} q_j}$ which are going to be use as reference. After the rough allocation is completed we are able to calculate $c_j = \frac{A_j}{\sum_{j=1}^{n} A_j}$ as the current coefficients at this stage;

2. we calculate the lag between the current values and the references, $\Delta_j = c_j - s_j$ ($j=\overline{1,n}$); $0 \le c_j \le 1; 0 \le s_j \le 1$ ($j=\overline{1,n}$), and the objective function is given by $f_{obj} = \min\left(\sum_{j=1}^{n} |\Delta_j|\right)$; $|\Delta_j|$ being the absolute value of Δ_j;

3. we select the $\min(\Delta_j)$ and $\max(\Delta_j)$, determining in this way the column which has the highest deficit and the column which has the highest surplus, respectively; a quantity, equivalent to a *lot size*, is transferred from the column with highest surplus to the column with the highest deficit; the row, respectively the price at which this switch is accomplished, is determined based on the most important impact [2, 3, 4] that the swap may produce at each iteration of the algorithm (*greedy* strategy's essence), considering at each attempt the availability of the necessary quantity that has to be transferred; if, at this step, we are not able to find two different columns for satisfying the *min-max* condition, then the algorithm stops here, and the current solution is considered the best that we can reach;

4. repeat from step 1, using as current solution the allocation resulted after the quantity equivalent to a *lot size* was switched between the two chosen columns; we obtain a new value for the objective function f'_{obj}. The algorithm continues until the stop condition is reached: $f_{obj} - f'_{obj} < e$, $0 < e < 1$.

It has to be specified the fact that we record the pairs of the columns between which a transfer has occurred, the row (associated to the executed price) and the direction of the transfer, in order to not go forward and backward inside the space of the solutions, and for avoiding the cyclical traps. This approach is consistent with the *greedy* method's general nature, and secures the reach of an optimal solution, or a solution located in the very vicinity to an optimal one.

For the next two algorithms, the basic idea is retained. The way in which we define the objective function determines essentially the final solution, function of which criteria is more relevant for the real purpose (context).

B. The objective function based on the average price.

This algorithm puts a greater emphasis on having an average price, for each account, as close as possible to the general average price: $\overline{p} = \frac{\sum_{i=1}^{n}(E_i \times p_i)}{\sum_{i=1}^{n} E_i}$, computed for each product.

The following steps describe the essence of the algorithm:

1. for each account we calculate the average price at this stage: $\overline{p}_j = \frac{\sum_{i=1}^{n}(a_{ij} \times p_i)}{\sum_{i=1}^{n} a_{ij}}$, $j = \overline{1, m}$;
2. calculate the *deltas* as $\Delta_j = \frac{\overline{p}_j}{\overline{p}} - 1$ ($j = \overline{1, m}$) and the objective function as: $f_{obj} = \min\left(\sum_{j=1}^{n} |\Delta_j|\right)$; $|\Delta_j|$ being the absolute value of Δ_j;
3. take advantage from using the same *min-max* technique that we have used in the previous algorithm, for detecting the pair of columns which will be involved in a swap of a *lot size*; the same stop condition for this step;
4. repeat from step 1 using as current solution the allocation resulted after the quantity equivalent to a *lot size* was switched between the two chosen columns; we obtain a new value for the objective function f'_{obj}. The algorithm continues until the stop condition is reached: $f_{obj} - f'_{obj} < e$, $0 < e < 1$.

C. The objective function based on the total value.

This variant of the algorithm combines the previous ones, offering the best balance between the fairly allocated quantities (based on the orders), and the desired average price for each client account as close as possible to the general average price $\overline{p}$.

1. At the first step we calculate the values allocated on client accounts after the *rough allocation* is completed, using the average price for each account determined as in the previously presented algorithm: $V_j = A_j \times \overline{p}_j$ ($j = \overline{1, n}$); the coefficients c_j become weighted averages: $c_j = \frac{A_j \times \overline{p}_j}{V} = \frac{V_j}{V}$ ($j = \overline{1, n}$), where $V = \sum_{j=1}^{n} V_j$;
2. calculate the *deltas* as $\Delta_j = c_j - s_j$ ($j = \overline{1, n}$); $0 \le c_j \le 1$; $0 \le s_j \le 1$ ($j = \overline{1, n}$), and the objective function is given by $f_{obj} = \min\left(\sum_{j=1}^{n} |\Delta_j|\right)$; $|\Delta_j|$ being the absolute value of Δ_j; the s_j are the initial, referential coefficients, used in the first presented algorithm;
3. use the same *min-max* strategy for determining the two columns which will be involved in the swap, with the same stop condition;
4. repeat, similarly from step 1; a new value for the objective function is obtained - f'_{obj}.

The algorithm continues until the stop condition is reached: $f_{obj} - f'_{obj} < e$, $0 < e < 1$.

The last algorithm provides better, overall, solutions. Function of the concrete demands, the first two algorithms may be more suitable for certain cases.

The algorithms are designed to solve the allocation problem for both *program trading* and *single stock trading*.

References:

[1] – George Nemhauser, Laurence Wolsey – *Integer and Combinatorial Optimization* – John Wiley & Sons, Inc., 1999;

[2] – Donald L. Kreher, Douglas R. Stinson – *Combinatorial Algorithms – Generation, Enumeration and Search* – CRC Press LLC, 1999;
[3] – William J. Cook, William H. Cunningham, William R. Pulleyblank, Alexander Schrijver – *Combinatorial Optimization* – John Wiley & Sons, Inc., 1998;
[4] – Zbigniew Michalewicz, David B. Fogel – *How to Solve It: Modern Heuristics* – Springer-Verlag Berlin Heidelberg, 2000.