American Journal of Operations Research, 2016, 6, 387-400
http://www.scirp.org/journal/ajor
ISSN Online: 2160-8849
ISSN Print: 2160-8830

Parallel *Quasi* Exhaustive Search of Optimal Asset Allocation for Pension Funds

Massimo Bernaschi, Mauro Carrozzo, Matteo Lulli, Giacomo Piperno, Davide Vergni

Istituto Applicazioni del Calcolo, Consiglio Nazionale delle Ricerche, Rome, Italy
Email: m.bernaschi@iac.cnr.it, m.carrozzo@iac.cnr.it, m.lulli@iac.cnr.it, giacomo.piperno@gmail.com, d.vergni@iac.cnr.it

How to cite this paper: Bernaschi, M., Carrozzo, M., Lulli, M., Piperno, G. and Vergni, D. (2016) Parallel *Quasi* Exhaustive Search of Optimal Asset Allocation for Pension Funds. *American Journal of Operations Research*, **6**, 387-400.
http://dx.doi.org/10.4236/ajor.2016.65036

Received: July 15, 2016
Accepted: September 10, 2016
Published: September 13, 2016

Abstract

We present a solution based on a suitable combination of heuristics and parallel processing techniques for finding the best allocation of the financial assets of a pension fund, taking into account all the specific rules of the fund. We compare the values of an objective function computed with respect to a large set (thousands) of possible scenarios for the evolution of the Net Asset Value (NAV) of the share of each asset class in which the financial capital of the fund is invested. Our approach does not depend neither on the model used for the evolution of the NAVs nor on the objective function. In particular, it does not require any linearization or similar approximations of the problem. Although we applied it to a situation in which the number of possible asset classes is limited to few units (six in the specific case), the same approach can be followed also in other cases by grouping asset classes according to their features.

Keywords

ALM, Pension Fund, Optimization, Parallel Processing

1. Introduction

A pension fund is a common asset pool run by a financial intermediary on behalf of a company and its employees, with the goal of generating stable growth over the long term and providing pensions for the employees when they retire [1]. Pension funds control relatively large amounts of capital and represent the largest institutional investors in many countries. At times, a pension fund provides a fixed, preset benefit for employees upon retirement, helping workers plan their future spending. In other cases, the pension received in the future by each participant depends on the actual return of the fund investments (fixed income, stocks, real estate, etc.). Regardless of the rules that specify the value of each pension, there is a significant dependence on several actuarial factors for the total amount of yearly liabilities of any pension fund. Once combined with the stochastic nature of the returns coming from the financial assets of the pension fund, this feature makes it necessary to have tools supporting decisions of pension fund management about assets allocation [2]. For any pension fund, the bare-minimum target of Asset and Liability Management (ALM) is to generate a stable cash flux in order to guarantee pension payments for a very long planning horizon [3]. To that purpose, ALM requires a risk analysis that, taking into account assets, liabilities and all the other features of any specific fund, can help the board of a pension fund in finding the most suitable investment policy [4] [5]. Within this context, we present a solution we developed for the pension fund of the personnel of a major Italian bank. The solution includes a software procedure that provides a close-to-optimal assets allocation policy with respect to a large set of future financial scenarios taking into account also the effects of different actuarial scenarios. The underlying model includes all the details of the specific pension fund so its results can be easily interpreted and used in practice. However, our approach to the optimization may be applied to a wide range of real world situations under the only condition that the number of asset classes among which the financial assets may be divided into is limited, or at least can be approximated, to few units. Moreover, to obtain a good approximation of the actual optimal solution, the objective function should have a smooth dependence on the portfolio composition so that tiny changes should never produce dramatic changes in the value of the objective function, although linearity is not necessary. The paper is organized as follows: Section 2 presents the model for the management of the pension fund; Section 3 describes the structure of the software solution we have developed for simulating the model; Section 4 proposes our approach to the optimization of the financial assets allocation based on a suitable combination of heuristics and parallel processing techniques; Section 5 concludes the work providing future directions of activity.

2. The Model

In this Section, we present the mathematical formulation of the asset and liability management problem for the pension fund. The model includes two sets of variables: the first one related to the actuarial component of the problem and the second one related to the financial component. First we describe, via coarse-grained variables, the actuarial aspects of the model, then we carefully depict the financial ones.

The output of the optimization problem is the allocation of the financial components of the fund assets that optimizes the expected utility function verifying a set of constraints, with respect to a wide set of different scenarios for the evolution of the stochastic variables.

2.1. Notation

Let T denote the optimization period, $T = \{t_1, \cdots, t_f\}$. For each $t \in T$ we define, for the actuarial variables:

- $a(t)$: total amount of pension contributions paid by members into the Fund at time *t*;
- $p(t)$: total amount of pensions received by retirees at time *t*.

The variables $a(t)$ and $p(t)$ represent the expected average evolution of the actuarial model that must consider all the possible events that may happen to each single participant in the fund. It is important to highlight that $p(t)$, for $t \in T$, depends on the decisions made by the Fund management, since the pensions vary according to the financial performance of the fund, as well as on actuarial events like death, invalidity, *etc.* of the participants [2]. As a consequence, it is not possible to separate the evolution of the actuarial component from the evolution of the financial component.

The assets of the Fund are invested mainly in three areas: rented real-estate properties, Italian government securities and a combination of financial instruments including stocks, corporate bonds, etc.

Currently the management of the real-estate properties and the trade of government securities are carried out following pre-defined (and fixed) internal policies. In the model, for the sake of simplicity, the disposal of the investment property and the trade of government securities are considered as input data used only for accounting reasons with no impact on the optimization process.

Let us define the following quantities:

- $c_R(t)$: capital invested in rented properties at time $t \in T$;
- $r_R(t)$: revaluation rate of capital invested in rented properties at time $t \in T$;
- $y_R(t)$: yearly income received as rent at time $t \in T$;
- $c_B(t)$: capital invested in government securities at time $t \in T$;
- $y_B(t)$: coupons received from government securities at time $t \in T$.

The allocation among other financial instruments is the only way to influence the yearly performance of the Fund. Hence, defining *K* as the set of financial assets whose allocation can be modified, we need to consider, for $k \in K$, the following quantities:

- $k \in K, n_k(t)$: number of shares in the *k*-th instrument at time $t \in T$;
- $k \in K, q_k(t)$: unit value of shares in the *k*-th instrument at time $t \in T$.

2.2. Variable Evolutions

Every year, the Fund collects contributions from active workers and pays pensions to retirees. We define the *actuarial deficit* or *surplus* as:

$$\delta(t) = a(t) - p(t). \tag{1}$$

Furthermore, the Fund receives rent and coupons from real estate properties and government bonds respectively. Hence, the overall *deficit* or *surplus* is given by:

$$\Delta(t) = \delta(t) + y_R(t) + y_B(t). \tag{2}$$

For the sake of simplicity, let us define the total value of financial instruments at time *t* as:

$$Q(t)=\sum_{k\in K} n_k(t)q_k(t), \tag{3}$$

and the yearly gain from them as:

$$X(t)=Q(t)-Q(t-1). \tag{4}$$

The nominal *rate of return* representing the Fund performance is defined as:

$$R(t)=\frac{\Pi(t)}{A(t-1)+0.5\delta(t)}, \tag{5}$$

where

$$A(t)=c_R(t)+c_B(t)+Q(t) \tag{6}$$

is the total fund capital at time t and

$$\Pi(t)=c_R(t-1)r_R(t)+y_R(t)+y_B(t)+X(t) \tag{7}$$

is the total gain from that capital.

Basically, the rate of return defined by (5) measures the ability of the Fund to yield a return provided the capital at the previous time while still considering the actuarial deficit.

According to the Fund's statute, the yearly contribution that each active worker pays into the Fund is a fraction of her/his salary which in turn evolves according to some corporate policy considering her/his category (director, manager, employee) and her/his length of service.

Particular attention must be paid to pensions that members receive from the Fund, since, according to the Fund rules, those strongly depend on the rate of return at the end of the year preceding the time the pension is received. Based on the contributions a member paid into the Fund before retiring, her/his *base pension* is computed: each year the retiree receives a *total pension*, $p(t)=I(t)p_b$, that is her/his base pension, p_b, times a given coefficient $I(t)$ which, in turn, is a function of the rate of return (5):

$$I(t)=\left[1+R(t)-C\right]I(t-1), \tag{8}$$

where C is a constant value defined in the Fund's statute.

Further details of the rules according to which the contributions and pensions are computed every year are not very interesting for the purposes of the present work; nevertheless, it is important to remind once more that $p(t)$, for $t\in T$, depends on the decisions the Fund makes as well as on actuarial events like death, invalidity, *etc.* of the participants: hence, the actuarial component of the model is strongly coupled with the financial one.

2.3. Allocation Strategy

Every year, the portfolio composition over *K*, the set of financial instruments, can be modified by buying/selling shares in order to fulfill the *balance constraint* (see next Section) or for market reasons (e.g., to take advantage of changed market conditions).

Let us define the percentage of allocation of the instrument $k\in K$ as:

$$x_k(t) = \frac{n_k(t)q_k(t)}{Q(t)}. \tag{9}$$

Variables $x_k(t)$ are the *control* variables, since at each decision step, a different proportion of $x_k(t)$ can be imposed, modifying the portfolio composition, albeit under the condition that a set of constraints is fulfilled.

2.4. Balance Constraint

The most important constraint the Fund has to fulfill is the *balance constraint*: the yearly actuarial deficit has to be covered by using the total gain from fund's investments or, if this is not possible, and under particular conditions, by selling shares of the financial assets.

The balance constraint can be written as:

$$\delta(t) + \Pi(t) \geq 0. \tag{10}$$

In order to get some insight into the previous equation we rewrite it as

$$\delta(t) + c_R(t-1)r_R(t) + y_R(t) + y_B(t) + \sum_{k \in K} n_k(t)q_k(t) = \sum_{k \in K} n_k(t-1)q_k(t-1)$$

where we use Equations (3), (4) and (7) and the identity sign. If the quantity $\delta(t) + c_R(t-1)r_R(t) + y_R(t) + y_B(t)$ is less than zero, then enough shares must be sold in order to balance out the deficit.

Other important constraints regard the limitations on the changes that can be introduced in the portfolio composition in a single decision time. These constraints will be presented in detail in Section 4. Basically they prevent from selling (or buying) a number of shares that exceeds a predefined percentage.

2.5. Objective Function

Given the rate of return in Equation (5), the Fund management defined the following objective function:

$$CF(t) = \sum_{k=3,10,30} \gamma_k \left| \rho_k \left[A(k-1) + 0.5\delta(k) \right] - \Pi(k) \right|, \tag{11}$$

where, for $k = 3, 10, 30$:

- $\gamma_k \in [0,1]$ represents the different weight that the different maturities have in the objective function, with $\sum_{k=3,10,30} \gamma_k = 1$;
- ρ_k are arbitrary constants representing targets on the return.

In words, once decided the return targets ρ_k at different periods, the management of the Fund has to select the best policy in order to achieve the expected return target at that given period.

3. Software Architecture

The model presented in Section 2 has been implemented in a software following a modular approach (see Figure 1) which makes it easier to evaluate different simulators for the evolution of the financial assets.

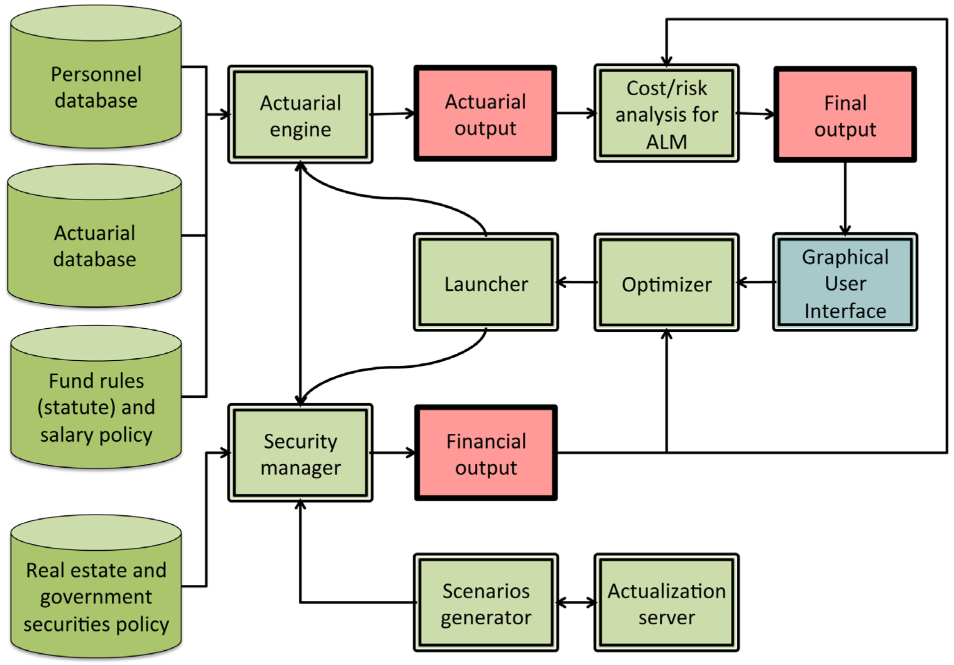


Figure 1. The software organization of the solution for Pension Fund ALM. Blocks in dark green represent input data; blocks in light green are computational engines; blocks in red are output data. The block in blue is the Graphical User Interface.

Each module is responsible for a particular part of the model. In this way, it becomes easier also to unit-test the individual components.

Actuarial engine: The total amount of contributions members pay into the Fund and the pensions received by retirees each year depend on particular events that may happen to members (death, invalidity, etc.) due to their characteristics and some given probabilities (reported in actuarial tables), and on the rules defined in the Fund's statute. In particular, a very important feature is that the pensions depend on the return of the Fund.

The *actuarial engine* is the software module dealing with statistical events and rules imposed by the Fund's statute.

The module communicates with the *security manager* by means of a simple textual protocol similar to HTTP: this protocol allows the actuarial engine to query the security manager for the rate of return and send the total amount of pensions paid to retirees and contributions received by members back to the former.

The ordered sequence of statistical events happening to members and retirees can be considered an *actuarial scenario*, and it is what we call a *repetition*.

Scenarios generator: The goal of the Fund management is to find the best strategy for the allocation of the capital among the available financial instruments over the simulation period.

In order to perform a cost-risk analysis that supports investments decisions, we need to test possible investment strategies on a set of scenarios, each representing the dynamics of $q_k(t)$, for any $k \in K$ and $t \in T$.

The *scenario generator* is in charge for producing such evolution, according to any of several possible statistical/probabilistic models. It is important to realize that the architecture of our solution does not depend on a specific model of assets' evolution. The model can be either a simple VAR with no specification of the financial meaning of each component or a classic model that describes the evolution of few basic financial/economic variables (e.g., risk-free interest rate, inflation, output gap, risk-premium, etc.) and then determines the value of the assets according to their relation with those fundamental variables.

Actualization server: This is a simple generator of the term structure used for the actualization of cash flows for the exposed *funding ratio*, *i.e.,* the ratio between the assets of the fund, including the discounted value of the future contributions, and the sum of the discounted value of the future pension payments. The term structure is generated according to the well-known Nelson-Siegel model [6] calibrated on a time frame of three-five years.

Security manager: This is the module that implements most part of the logic described in the present work. The *security manager* queries the scenario generator and the actualization server for data to test the chosen investment strategy on.

For each scenario, and for each repetition on that scenario, the security manager computes all the variables required for the evaluation of the objective function. Such dataset is further processed to carry out a complete statistical analysis that is finally presented to the user.

Launcher: The Investor must be confident as much as possible about her/his decisions, thus each strategy has to be tested on a pretty large set of scenarios (at least 5000) and a reasonably large set of repetitions on each scenario (~20). If S is the set of scenarios and R is the set of repetitions, the dimension of the problem is $O(|S|\cdot|R|)$.

Given the number of participants in the Fund, the computation of many statistical events (depending on a large set of random numbers) all over the simulation period may be a very demanding process from a computational point of view. Hence it is very important to parallelize independent streams of computation.

Since both the scenarios and the repetitions are based on pseudo-random numbers, it is easy to reproduce a certain scenario or a certain repetition provided we know the corresponding random number generator seed.

The *launcher* is responsible for launching different instances of the previous modules and collect their output once they terminate.

Chart generator: When the launcher ends, the results can finally be presented as a set of plots. This module parses the output files and generates various kinds of charts that the user can examine through the Graphical User Interface. Although the main purpose of the present work is not the visualization of the results, we report a sample output in **Figure 2**.

Graphical User Interface: The purpose of this module, based on the GTK library, is to let the user conveniently insert data in a human readable form or through Excel files, run the launcher and present the tabular/graphical output.

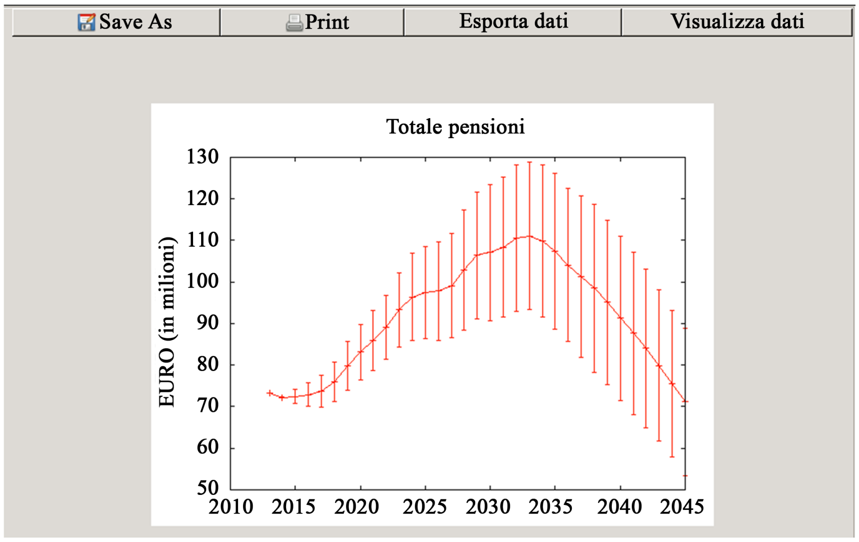


Figure 2. Example of output of the model: total amount of pensions paid in a simulation spanning years 2013-2045. The picture shows that it is possible to print the plot, save the results in plain text format and export them to Excel.

Software technologies: The software is highly portable and may run under the control of Linux, Mac OS X and Windows Operating Systems. The computational engines (*i.e.*, the blocks in light green in Figure 1: *optimizer*, *scenarios generators*, etc.) are written in C and C++. Modules exchange data each other by using *stream sockets*. This makes possible to run the modules on distinct systems. Parallel processing exploits multi-process techniques. Due to the differences between Linux/Mac OS X and Windows this part is operating system dependent. We do not resort to multi-threading since the computation done by each task takes so much time that the overhead due to tasks creation (that is, regardless of the operating system, much higher than the overhead due to thread creation) is negligible. Moreover, a multi-task implementation can be ported to a distributed platform in a much easier way.

4. The Optimization Process

In this Section we present our approach to the optimization of the objective function of the model presented in Section 2.

Given the initial composition of the investment portfolio divided into N asset classes, the goal is to determine, through the optimization process, the best allocation strategy with respect to the given objective function. With the term *strategy* we mean a time series of N-tuples, one for every year over the specified simulation period. For a given year, the corresponding N-tuple contains the N percentage values expressing the relative proportion of the N asset classes on the total investments (*i.e.*, the control variables defined in (9)).

It is apparent that the objective function has a critical role in choosing the optimization process. In general, nonlinear objective functions as well as nonlinear constraints may turn an optimization problem into a computationally intractable one.

Here, we propose an approach that may be used when nonlinear computationally intensive optimization problems are encountered. In our case, we have an objective function that makes it possible to exploit parallel computing architectures: the computations required for the evaluation of S scenarios are divided onto the C cores available on any modern CPU and among the CPUs of multiprocessor systems.

We assume that the following constraints hold for the optimization process:

•for any asset class, the relative percentage may vary only within a given asset class specific range;

•for any asset class, the possible allocation change cannot exceed its specific turnover limit;

•the sum of the relative percentages for all asset classes must always add up to 100%.

We also assume that the N asset classes can be partitioned into two groups: G_1 with N_1 asset classes and G_2 with N_2 asset classes ($N_1 + N_2 = N$), where the assets in G_1 account for the prevailing amount of the initial portfolio allocation (e.g., 2/3 of the total capital). In general, the portfolio can be partitioned into more than two groups: in this case the optimization steps (see below) can be applied in sequence to all groups. In other words, we introduce a hierarchy among the asset classes, assuming that the optimum can be approximated at the leading order by varying the composition of the first (most significant) G_1 group alone whereas other variations would lead to sub-leading corrections. For objective functions depending in a *balanced* way on the portfolio composition this assumption is fully acceptable since it is very unlikely that any asset class having a limited weight had a dramatic impact on the value of any *reasonable* objective function.

The search for a quasi-constant allocation strategy is a further simplification in our optimization procedure. In this respect, we consider *constant* an allocation strategy where the relative percentages of all asset classes remain constant throughout the whole simulation period. A quasi-constant allocation strategy is a constant strategy that may be modified only in a predefined set of revision years selected by the Investor, *i.e.*, for $t \in T$. Within this framework, the optimization process proceeds as follows:

Step 1: Exhaustive search on G_1. The above mentioned constraints define a set S_1 of N_1 -tuples: we search the best constant allocation strategy for the asset classes in G_1 by evaluating the objective function for all possible combinations in S_1 while keeping the allocation in G_2 equal to the initial one. Once the best constant strategy is found, the allocation of the assets in G_1 is linearly transformed into the best one, starting from the initial year using the minimum number of years compliant with turnover limits[1].

Step 2: Exhaustive search on G_2. Given the set S_2 of N_2 -tuples as defined by the above mentioned constraints, we search the best constant allocation strategy for the assets in G_2 by evaluating the objective function for all possible combinations in S_2 while keeping the allocation in G_1 equal to that computed in Step 1. Once the best

[1]E.g., if the optimal percentage for an asset class is 27%, the current percentage is 24% and the turnover limit is 1.5%, then two years are required to transform the current percentage in the optimal one.

constant strategy is found, the combination of the asset classes in G_2 is linearly transformed into the best one following the same procedure described in Step 1.

The partitioning into two sets of asset classes affords a remarkable reduction in computing requirements. If m_i is the number of possible percentages for the i^{th} asset, performing the exhaustive search on N asset classes requires the computation of $m_1 \times m_2 \times \cdots \times m_N$ evaluations of the objective function whereas, when the assets are partitioned into two groups, we need to compute only $m_1 \times \cdots \times m_{N_1} + m_{N_1+1} \times \cdots \times m_{N_1+N_2}$ evaluations. For instance, if we have $N = 6$ asset classes partitioned into two groups ($N_1 = N_2 = 3$) and $m_i = 10$ for all asset classes, we have to evaluate $1000 + 1000 = 2000$ times the objective function instead of 1,000,000 times (the work is reduced to 0.2% of the original one). Furthermore, if we define

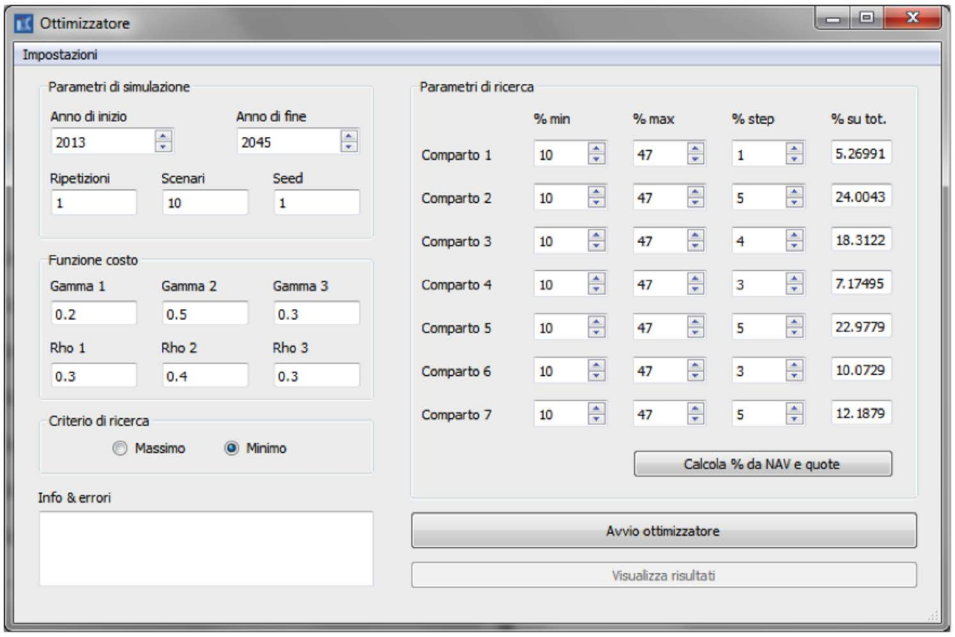
$$\mathrm{Sum}_{G_1} = c0_1 + \cdots + c0_{N_1} \quad \text{and} \quad \mathrm{Sum}_{G_2} = c0_{N_1+1} + \cdots + c0_N = 100 - \mathrm{Sum}_{G_1}$$

where $c0_i$ ($i = 1, \cdots, N$) are the initial percentages of the asset classes, the constraints imply that (within a group) the sum of asset classes percentages is constant and equal to Sum_{G_j} ($j = 1, 2$). This constraint further simplifies the optimization problem since only $N_j - 1$ asset classes percentages vary independently within G_j whereas the last N_j is given by the difference with respect to Sum_{G_j}.

Step 3: Optimality check on revision years for group G_1. For any year in which a revision of the allocation is possible (or strictly required), the objective function is evaluated on a set of new strategies obtained from the current one by varying (locally, within the turnover limits) the allocation from the current revision year up to the end of the simulation period. If a better allocation strategy is found, that becomes the new quasi-constant allocation strategy.

Step 4: Optimality check on revision years for group G_2. This is analogous to what is done in Step 3 for the asset classes in G_1.

By using the GUI shown in **Figure 3** and **Figure 4** the user defines the constraints of the optimization process, the number of scenarios and the initial composition of the portfolio. The optimization process is carried out by running a subset of the scenarios evaluations on each of the available cores. When all scenarios have been evaluated, the average value of the objective function is determined and compared with the current reference (*i.e.*, best) value. If the new value is better (*i.e.*, lower for the objective function defined by Equation (11)), it replaces the reference value and the asset allocation that produced it becomes the new reference asset allocation. Then a new asset allocation is selected according to the rules above explained and the procedure starts to evaluate its performance with the same set of scenarios. Note that the parallelization is carried out so that everything but the subset of scenarios is the same for all cores running the optimization procedure. Moreover there is no need to exchange data among the different instances during the evaluation and in the end only the value of the objective function must be communicated. As a matter of fact, our performance tests with increasing number of cores (up to 48) and a fixed number of scenarios and repetitions (5000 and 10, respectively) showed a close-to-ideal speedup (*i.e.*, linear with respect to the number of cores). To further speedup the optimization


Ottimizzatore

Impostazioni

Parametri di simulazione

Anno di inizio	Anno di fine	
2013	2045	
Ripetizioni	**Scenari**	**Seed**
1	10	1

Funzione costo

Gamma 1	Gamma 2	Gamma 3
0.2	0.5	0.3
Rho 1	**Rho 2**	**Rho 3**
0.3	0.4	0.3

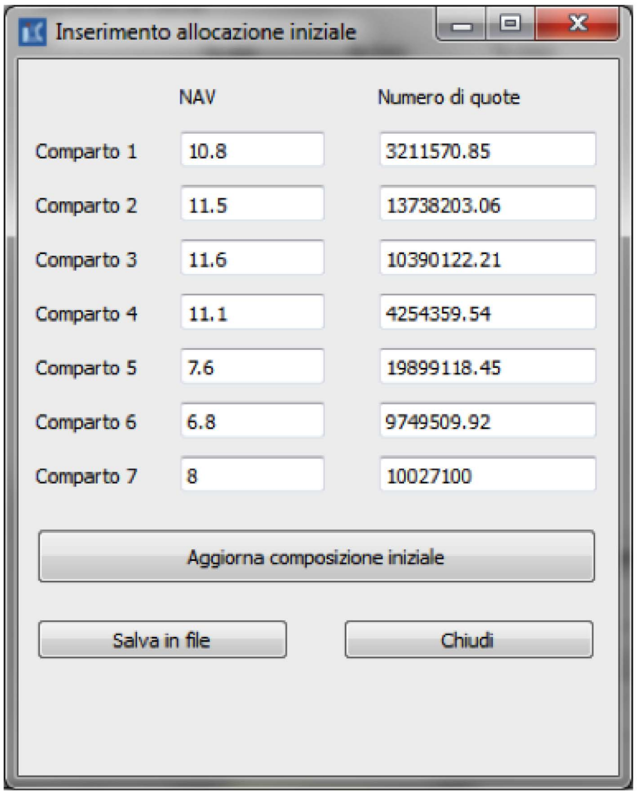
Criterio di ricerca: Massimo [✓ Minimo]

Info & errori

Parametri di ricerca

	% min	% max	% step	% su tot.
Comparto 1	10	47	1	5.26991
Comparto 2	10	47	5	24.0043
Comparto 3	10	47	4	18.3122
Comparto 4	10	47	3	7.17495
Comparto 5	10	47	5	22.9779
Comparto 6	10	47	3	10.0729
Comparto 7	10	47	5	12.1879

Figure 3. The GUI by which the user can set up the optimization procedure. In particular it is possible to define lower and upper bounds for the percentages of each asset classes (in this case there are seven asset classes) and the turnover limit for each decision time.


Inserimento allocazione iniziale

	NAV	Numero di quote
Comparto 1	10.8	3211570.85
Comparto 2	11.5	13738203.06
Comparto 3	11.6	10390122.21
Comparto 4	11.1	4254359.54
Comparto 5	7.6	19899118.45
Comparto 6	6.8	9749509.92
Comparto 7	8	10027100

Figure 4. The GUI by which the user can set up the initial number of shares for each asset class and the initial Net Asset Value (NAV) of the single share. Future NAVs for each asset class are determined by the *scenarios generator*.

procedure, during the evaluation of the scenarios, only the quantities required for the computation of the objective function are saved. When the optimization procedure completes, the user can load the resulting asset allocation generating all other quantities of interest as explained in Section 3.

Figure 5 shows what we call *search surface* for a typical execution of our optimization procedure applied to the model described in Section 2. There are three asset classes in the first group G_1: c_2 (government securities), c_3 (high yield corporate bonds) and c_5 (stocks). The initial (*i.e.*, the pre-existing) allocation (see also **Figure 3**) is $c_2 = 24\%$, $c_3 = 18.3\%$ and $c_5 = 23\%$ of the total amount of financial assets. The optimization procedure with *default* constraints on lower (10%) and upper (47%) bounds and turnover limit requires a total of 90 iterations (52 for Steps 1 - 2 and 38 for Steps 3 - 4) using 5000 scenarios and 20 repetitions for the actuarial factors. Total execution time on a server equipped with 4 Intel Xeon E5645 running at 2.4 GHz providing a total of 24 cores and 96 GB of RAM is slightly less than 12 hours (we do not use hyper-threading). The final (best) allocation for the three asset classes is: $c_2 = 25\%$, $c_3 = 30\%$, $c_5 = 11\%$.

5. Summary and Conclusions

We presented an alternative to the widely used techniques of stochastic programming

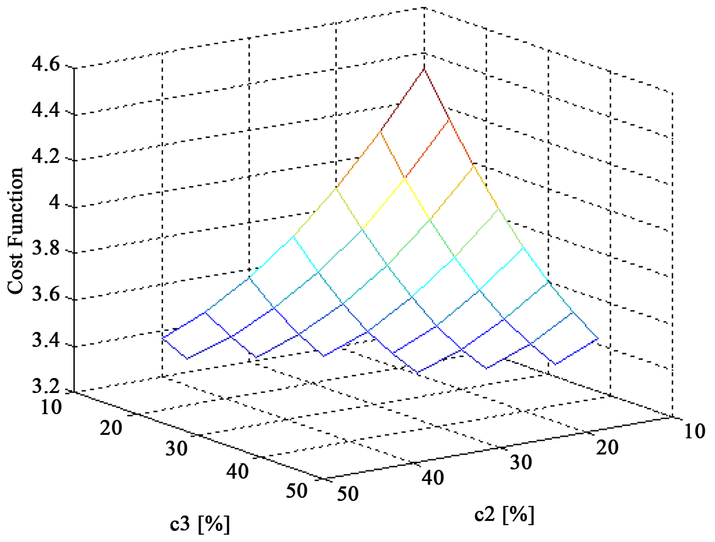


Figure 5. Search surface for asset classes c_2 and c_3 (part of G_1, the first group of asset classes). The x axis shows the percentage of asset class c_2 (Governments Securities) whereas the y axis shows the percentage of asset class c_3 (High Yield Corporate Bonds). The percentage of c_5, the last asset class in the G_1 group (stocks) is computed as difference between the total percentage of group G_1, that is assumed to be constant, and the sum of the percentages of c_2 and c_3. The z axis shows the value of the cost (or objective) function defined by Equation (11). For this objective function, lower is better, and the best asset allocation for the G_1 group found by the optimization procedure is $c_2 = 25\%$, $c_3 = 30\%$, $c_5 = 11\%$, (the total percentage of group G_1 is ~66%).

[7] for the optimization of the financial assets allocation for a pension fund. The asset classes are divided in groups and then the exploration of the space of possible solutions is carried out in two phases: a first exhaustive search of the optimal initial allocation and a second local (around the initial optimal allocation) search for possible adjustments in later (pre-set) times. There are several advantages following this approach: the main being the possibility of using large sets of different scenarios for the evolution of the Net Asset Value of the asset classes without imposing any *a priori* structure like scenario-trees. This fact has a significant relevance because with six asset classes the size of a decision tree becomes quickly too large. If we indicate with T the number of decision stages and with c the number of possible states for the NAV of any asset[2], the number of branches increases as $\left(c^6\right)^T$ so that even $T = 3$ becomes unfeasible. The second advantage is its flexibility that makes it possible to take into account all the details of the real-world problem under study. For instance, in the case of the asset allocation for the pension fund, the problem is not linear: there is a relation among the return of the fund, the pensions received by retirees and the total value of financial assets since, in case of balance problems, a suitable number of shares of the financial assets must be sold according to the Fund's statute. In situations like this, most portfolio selection models are simplified just to make them *solvable* by using standard techniques requiring limited computing resources (e.g., linear programming). However, the solution obtained with this simplification may be very far from the optimal one. With our approach, there is no limitation in the description of the problem imposed just for the sake of simplicity. The only assumption is that the objective function does not have too many extrema (minima or maxima depending on the formulation) very far from each other. The idea of using techniques of exhaustive search in combinatorial optimization by exploiting the potential of parallel processing is not brand new [8]. However, to the best of our knowledge, it has never been applied to asset allocation problems. Techniques of exhaustive search may be applied also to optimization problems in engineering design [9] [10], but in those cases, in general, there is no need to update the solution at different simulation times. Our *local search* around the optimal initial allocation makes that problem tractable under pretty reasonable assumptions. We expect to extend the present work along several possible directions. First of all, we are going to apply the same approach to similar problems of asset allocation and to extend the software architecture to support distributed evaluations of scenarios. We also expect to test alternative approaches like *randomized search* [11] to improve the performance of the periodic refinement of the solution. Finally, we want to assess whether a double-level parallelism (scenarios evaluation and multiple concurrent searches) may provide any advantage.

References

[1] Sharpe, W.F. (1976) Corporate Pension Funding Policy. *Journal of Financial Economics*, **3**, 183-193. http://dx.doi.org/10.1016/0304-405X(76)90002-7

[2]For instance if the NAV may remain the same, increase or decrease of a fixed percentage, then $c = 3$.

[2] Micocci, M., Gregoriou, G.N. and Masala, G.B. (2010) Pension Fund Risk Management: Financial and Actuarial Modeling. CRC Press, London.

[3] Boulier, J.F., Trussant, E. and Florens, D. (1995) A Dynamic Model for Pension Funds Management. *Proceedings of the* 5*th AFIR International Colloquium*, **1**, 361-384.

[4] Boender, C.G.E. and VOS, M. (2000) Risk/Return Budgeting at Pension Plans. *Institutional Investor Journal*, *Special Issues Fall* 2000, **1**, 80-88.

[5] Bogentoft, E., Romeijn, H.E. and Uryasev, S. (2001) Asset/Liability Management for Pension Funds Using CVaR Constraints. *The Journal of Risk Finance*, **3**, 57-71. http://dx.doi.org/10.1108/eb043483

[6] Nelson, C.R. and Siegel, A.F. (1987) Parsimonious Modeling of Yield Curves. *The Journal of Business*, **60**, 473-489. http://dx.doi.org/10.1086/296409

[7] Shapiro, A., Dentcheva, D. and Ruszcsyński, A. (2009) Lectures on Stochastic Programming. SIAM, Philadelphia.

[8] Nievergelt, J., Gasser, R., Maser, F. and Wirth, C. (1995) All the Needles in a Haystack: Can Exhaustive Search Overcome Combinatorial Chaos? In: van Leeuwen, J., Ed., *Computer Science Today*, Springer, Berlin, Heidelberg, 254-274.

[9] Roy, R., Hinduja, S. and Teti, R. (2008) Recent Advances in Engineering Design Optimisation: Challenges and Future Trends. *CIRP Annals—Manufacturing Technology*, **57**, 697-715.

[10] Parkinson, A.R., Balling, R.J. and Hedengren, J.D. (2013) Optimization Methods for Engineering Design. Brigham University Press, Provo.

[11] Solis, F.J. and Wets, R.J.B. (1981) Minimization by Random Search Techniques. *Mathematics of Operation Research*, **6**, 19-30. http://dx.doi.org/10.1287/moor.6.1.19