Università degli Studi di Modena e Reggio Emilia
Dipartimento di Economia Politica

Materiali di discussione

\\ 596 \\

Differential Evolution and Combinatorial Search for Constrained Index Tracking

by

Thiemo Krink[1]
Stefan Mittnik[2]
Sandra Paterlini[3]

Ottobre 2008

1 Group Investments, Allianz Investment Management SE (AIM), Königinstraße 28, 80802 Munich,GER

2 Center for Quantitative Risk Analysis, Dept. of Statistics, Ludwig-Maximilians- University Munich, Akademiestr. 1/I, 80799 Munich, Germany

3 Dept. of Economics, CEFIN & RECent, Univ.of Modena and Reggio E., Viale J. Berengario, 51, Italy
sandra.paterlini@unimore.it

Differential Evolution and Combinatorial Search for Constrained Index Tracking

Thiemo Krink[a], Stefan Mittnik[b], Sandra Paterlini[c,1]

[a]*Group Investments, Allianz Investment Management SE (AIM), Königinstraße 28, 80802 Munich,GER*

[b]*Center for Quantitative Risk Analysis, Dept. of Statistics, Ludwig-Maximilians-University Munich, Akademiestr. 1/I, 80799 Munich, Germany*

[c]*Dept. of Economics, CEFIN & RECent, Univ.of Modena and Reggio E., Viale J. Berengario, 51, Italy*

Abstract

Index tracking is a valuable low-cost alternative to active portfolio management. The implementation of a quantitative approach, however, is a great challenge from an optimization perspective: the optimal selection of a group of assets that can replicate the index of a much larger portfolio requires both to find the optimal asset positions and to fine-tune their allocation weights. The former is a combinatorial problem, whereas the latter is a continuous numerical problem. Both optimization problems need to be tackled simultaneously, because whether a selection of asset positions is promising or not depends on the actual allocations and vice versa. Moreover, the problem is usually high dimensional; typically an optimal subset of 30-150 positions out of 100-600 need to be selected and their asset allocation weights need to be determined. Search heuristics can be a viable and valuable alternative to traditional methods, which often cannot deal with the problem. In this work, we describe a new optimization method, which is partly based on Differential Evolution (DE) and on combinatorial search. The main advantages of our method are that it can tackle the index tracking problem as complex as it is while obtaining very accurate and robust results.

Key-words*: Index Tracking, Passive Asset Management, Differential Evolution, Combinatorial Search*

---

[1] Corresponding author: Sandra Paterlini, email: sandra.patelini@unimore.it, Tel. 00390592056848, Fax: 00390592056947

1 Introduction

Portfolio construction is a key issue in asset management. In recent years, benchmark replication by index tracking has become a popular and valuable low-cost alternative to active portfolio management. The implementation of a quantitative approach to tackle the problem of optimal benchmark replication, however, is far from trivial, both, from an econometric and an optimization perspective.

From an econometric viewpoint, the main problems consist in modeling the objectives such that they reflect the index tracking problem properly and in estimating/forecasting returns and covariance matrixes such that the tracking portfolio can replicate the index not only in sample but also out of sample. The reader is refereed to Beasley et al. (2003) for a literature survey on different approaches to index tracking.

Apart from the econometric issues, index tracking poses a challenging optimization problem. The optimal selection of a small set of assets that can replicate the index of a much larger portfolio requires both to find the optimal asset positions and to fine-tune their asset allocation weights. The former is a combinatorial problem, whereas the latter is a continuous numerical problem. Both optimization problems need to be tackled simultaneously, because whether a selection of asset positions is promising or not depends on the actual allocations and vice versa. Moreover, the problem is high dimensional. Typically an optimal subset of 30-150 positions out of 100-600 need to be selected and their asset allocation weights need to be determined.

If we choose the tracking error volatility as measure of goodness in index tracking and we know the index composition, such problem seems to be quadratic and it is tempting to assume that it can be solved with quadratic programming (QP). However, this is not even possible in combination with branch and bound for realistic problem instances since it would require evaluating an enormous number of possible asset position combinations. Furthermore, it often happens that there are non-linear constraints that need to be satisfied in a realistic context (e.g.: art.22: the concentration limits of the UCITS rules[2], which is binding for most portfolio in EU, which requires that the sum of all asset weights which are larger than 5% must be smaller than 40%).

A valuable alternative is to use optimization heuristics (also known as search heuristics), which iteratively refine candidate solutions to the problem. They have the advantage that they are by far more general in their scope of application, i.e., they can handle the problem as it is, whereas conventional techniques require rigid assumptions such as linearity of constraints and a linear or

[2] European Union Directive, UCITS 3 (Undertaking For Collective Investments in Transferable Securities).

quadratic objective function. Optimization heuristics can even be applied when the optimization problem cannot be stated formally by a set of equations. The only prerequisite is that there is some kind of method that can compare the quality of solutions, here the quality of asset weight vectors regarding the tracking error. A drawback, however, is that there is no guarantee that the final result is optimal within finite computation time. Moreover, the accuracy and speed of convergence towards the optimum depends on the characteristics of the optimization problem and the choice of the algorithmic parameters.

Perhaps the most popular search heuristics are genetic algorithms (GA) (Fogel et al. 1966, Holland 1975) and Simulated Annealing (SA) (Kirkpatrick et al. 1983). Such heuristics have already been applied to the index tracking problem (See Beasley et al. 2003 and references therein), supporting them as a valid tool in such context. Recent studies, however, have shown for a great number of different problem instances that GAs and SA are not good choices when tackling non-trivial continuous numerical problems. Instead Differential Evolution (DE) (Storn and Price, 1997) and evolution strategies (ES) (Rechenberg, 1973) should be used. DE has shown remarkable performance compared to other heuristics, such as Particle Swarm Optimization (Kennedy and Eberhart 1995) and Genetic Algorithm on continuous numerical problems (Vesterstrøm and Thomsen 2004, Paterlini and Krink. 2006, Krink et al. 2007). Research on index tracking (and on non-continuous numerical problem) using DE is still at an early stage, but recent results seem to promote its usage in such applications, even if further analysis is still required (Maringer and Oyewumi, 2007). In fact, even if DE is specialized on continuous, numerical optimization, we believe that they could be easily adapted to successfully tackle also combinatorial problem as the tracking index one. In this work, we propose a hybrid algorithm (henceforth: DECS-IT: Differential Evolution and Combinatorial Search for Index Tracking) that extends DE with an additional combinatorial search operator to determine the optimal choice of asset positions. Then, we show that DE can also deal successfully with non-continuous numerical problem. Furthermore, since we want to the tackle the index tracking problem in a real-world context, we consider some of the main real-world constraints a manager would face and implement a constraint handling routine using a combination of techniques, which are specifically tailored to the index tracking problem.

Empirical results show that the main advantages of our method are that it can tackle the index tracking problem as complex as it is while obtaining very accurate and robust results.

The remaining sections of this paper are organized as follows. In section 2, we will give a formal definition of the index tracking problem as an error minimization problem. Section 3 outlines our novel optimization algorithm and in section 4 we present results on seed initialization and on a comparative study with quadratic programming regarding a real world dataset. Section 5 describes a realistic financial application. Finally, we summarize and discuss our results in section 6.

2 The Index Tracking Optimization Problem

In this study, we state the optimization goal of the index tracking problem as searching the optimal portfolio with respect to minimization of tracking error volatility of a benchmark (index), defined as:

$$\underset{\mathbf{w}}{minimize} \quad f(\mathbf{w}) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(R_t^{\,P} - R_t^{\,B})^2} \tag{1}$$

subject to

(1) $\sum_{i=1}^{n} w_i = 1$

(2) $0 \le w_i \le 1$

(3) $\varepsilon_i \delta(w_i) \le w_i \le \xi_i \delta(w_i), \delta(w_i) = \begin{cases} 1 & \text{if } w_i > 0 \\ 0 & \text{otherwise} \end{cases}$

(4) $L \le \sum_{i=1}^{n} \delta(w_i) \le K$

(5) $\sum_{i:w_i > Lb} w_i \le Ub$

(6) $\mathbf{Aw} \le \mathbf{b}$ and $(\mathbf{A}_{eq}\mathbf{w} = \mathbf{b}_{eq})$

where

$t=1,...,T$	the time period
n	the number of available assets (composing the index)
$R_t^{\,B} = \ln(\frac{B_t}{B_{t-1}})$	the index (benchmark) log-return at time t
B_t	the index value at time t
$R_t^{\,P} = \mathbf{rw}$	the portfolio return at time t
$\mathbf{r}_{txn}=[r_{t,i}]$	the txn vector of asset log-returns at time t
$r_{t,i} = \ln(\frac{S_{t,i}}{S_{t-1,i}})$	the i-th asset return at time t
$S_{t,i}$	the stock price of the i-th asset at time t (i=1,...,n)
w_i	the portfolio weight $0 \le w_i \le 1$ of the ith asset
$\mathbf{w}_{nx1}=[w_1,....,w_n]$	the tracking portfolio weights vector
$\mathbf{wb}_{nx1}=[wb_1,\ldots,wb_n]$,	the benchmark(index) portfolio weights vector

ε_i, ξ_i lower and upper bounds for each individual asset weight

$$\delta(w_i) = \begin{cases} 1 & if\ w_i > 0 \\ 0 & otherwise \end{cases}$$

L, K lower and upper bounds for the number of asset positions in the replication

Lb lower bound threshold for characterizing asset weights as "very large"

Ub maximal percentage of the sum of "very large" asset weights

$\mathbf{A}_{n.ineq.constraints\ x\ n}$ matrix of inequality constraints (e.g.: sector weight stability, group limit, avoid negative bias toward small cap stock)

$\mathbf{b}_{n.ineq.constraints\ x\ 1}$ vector of inequality constraints

$\mathbf{A}_{\mathbf{eq}\ n.eq.constraints\ x\ n}$ matrix of equality constraints

$\mathbf{b}_{\mathbf{eq}\ n.eq.constraints\ x\ 1}$ vector of equality constraints

If the benchmark composition, **wb**=[wb_1**,…,** wb_n], is known and constant over the time period, we could re-state the index replication goal as to obtain an optimal portfolio (**w**=[w_1**,…,** w_n]) with respect to minimization of tracking error volatility of a benchmark (**wb**), such that:

$$\underset{\mathbf{w}}{minimize}\ f(\mathbf{w}) = (\mathbf{w} - \mathbf{wb})^T \boldsymbol{\Sigma} (\mathbf{w} - \mathbf{wb}) \qquad (2)$$

where $\boldsymbol{\Sigma}$ is the asset covariance matrix.

Such specification could allow to disentangle further the econometric and optimization problem and to better test for different covariance matrix estimation approaches. We also use such objective function to compare DECS-IT algorithm with respect to QP in some test cases.

Furthermore, the DECS-IT algorithm can easily deal with other objective functions, such as (Beasley et al. 2003, Gilli and Kellezi, 2002):

$$\underset{\mathbf{w}}{minimize} \quad f(\mathbf{w}) = \lambda E_T - (1-\lambda) R_T \qquad (3)$$

$$\text{where} \quad E_T = \frac{\left(\sum_{t=0}^{T} \left| r_t^P - r_t^I \right|^{\alpha} \right)^{\frac{1}{\alpha}}}{T} \quad and \quad R_T = \frac{\sum_{t=0}^{T} \left(r_t^P - r_t^I \right)}{T}$$

Equation (3) allows to model in a single objective function the fact that managers could happily accept positive deviations from the benchmark. The problem could otherwise be re-formulated as a

multi-objective optimization problem and still be successfully tackled with DE (Krink and Paterlini, 2007).

In the case of the tracking index problem, the main challenge from an optimization viewpoint is usually not to deal with the objective function, which often is quadratic, but to deal with the asset position *selection* (constraint 4) and other non-linear constraints, such as the concentration limit of the UCITS rules (constraint 5, where Lb=5%, Ub=40%), which is binding for most portfolios in the EU (German Investment Law §60). Asset position *selection* means that only a certain number of asset positions of the benchmark is used in the replication. Formulated as constraint 4, this means that the number of non-zero weights is bound, which is an integer constraint and hence non-linear[3].

Note, that this problem is neither a quadratic programming problem, because of the non-linear constraints, nor a mixed integer problem, because the domain of all variables (w_i) is continuous. Mixed integer problems consist of some variables that are continuous and others that are integer. The problem is hard because of constraints (4) and (5), in particular the issue of (discrete) asset position selection while searching for (continuous) asset allocation weights, and its high dimensionality.

In principle, integer constraints, like constraint 4, can be tackled by applying QP and branch and bound search (BB) to a reformulated version of the problem. The idea is to divide the original problem into a set of quadratic programming sub problems and then to search efficiently for the best obtainable sub solution with BB. For each sub problem constraint 4 is replaced by constraints that define one (fixed) choice for the asset positions, i.e., the search for the optimal solution of the sub problem is only concerned in finding the optimal weights for one particular choice of asset positions. The search space for BB, however, would be gigantic for a realistic problem instance. For instance, for a maximum of 50 asset positions out of 225 in the benchmark, one would need to search for the optimal solution among all possible combinations of 50 positions in 225, which is equal to 6.4443e+052 QP sub problems.

Moreover, constraint 5 cannot even theoretically be tackled with BB, because it cannot be divided in a discrete number of sub problems.

Therefore, the combination of BB and QP is not applicable. Search heuristics, as the one we used in this study, are a less restrictive optimization approach that can tackle these issues.

[3] Integer constraints cause non-convexity of the problem (discrete jumps) and therefore also non-linearity.

3 The DECS-IT optimization heuristic

The optimization heuristic that we introduce in this paper, DECS-IT, consists of four main algorithmic aspects that we describe in the following subsections.

3.1 Differential Evolution and Index Tracking

Evolutionary algorithms have already shown to be a valuable tool when applied to index tracking problems (see Beasley et al. 2003 and references therein). Their main advantage is that they can tackle optimization problems as they are without requiring rigid properties, such as continuity, linearity or convexity of objective functions and constraints. Hence, they can easily deal with cardinality constraints, non-linear constraints and other real-world settings passive managers have often to face when they want to implement a quantitative approach to index tracking.

Differential evolution (Storn and Price, 1995, Price, 1999) is fairly novel population-based search heuristics which is simple to implement and requires little parameter tuning compared to most other search heuristics. Differential Evolution has shown superior performance, with respect to other heuristics, such as Genetic Algorithms and Particle Swarm Optimization, in dealing with single-objective and multi-objective, noise free, numerical optimization problems (e.g. Ursem and Vadstrup, 2003, Krink *et al.*, 2004, Vesterstrøm and Thomsen, 2004, Paterlini and Krink, 2006). Some recent studies suggest that such heuristic could effectively tackle multiobjective portfolio optimization (Krink and Paterlini, 2007) and index tracking problems (Dietmar and Oyewumi, 2007). We aim to contribute to the research in such direction and show how such heuristic combined with other operators can provide reliable and robust results to the index tracking problem. Differential evolution is a population based search heuristic. After generating and evaluating an initial population $\boldsymbol{P}$ of candidate solutions, the candidate solutions are iteratively refined as follows: First, for each candidate solution $\boldsymbol{P}(j)$, choose three other indexes k, l, and m randomly between 1 and the size of the population (with $j \neq k \neq l \neq m$). Create a new candidate solution $\boldsymbol{c}$ by multiplying the difference between $\boldsymbol{P}(k)$ and $\boldsymbol{P}(l)$ with a so-called scaling factor f and add the result to $\boldsymbol{P}(m)$, i.e.,

$$\boldsymbol{c} = f \cdot (\boldsymbol{P}(k) - \boldsymbol{P}(l)) + \boldsymbol{P}(m)$$

Second, substitute $\boldsymbol{c}$ by the result of a recombination, called: crossover, between $\boldsymbol{c}$ and $\boldsymbol{P}(j)$, such that for each component o of the candidate solution, $\boldsymbol{c}(o) = \boldsymbol{P}(j, o)$ with a probability of 1-cf, where cf is the so-called crossover factor.

Third, evaluate the new candidate solution with the fitness function and apply selection by substituting ***P(j)*** with ***c*** (and updating the fitness of ***P(j)***) in case that ***c*** has a better fitness.
This procedure closely resembles the so-called Rand/1/Exp DE operator. The process is repeated for a fixed number of iterations and the optimization result is the best recorded candidate solution and fitness at the end of the run.

3.2 Position swapping

The main challenge of the index tracking problem is to tackle the continuous numerical and combinatorial aspects for a very large number of parameters simultaneously.
The combinatorial problem consists of finding the optimal subset of asset positions to minimize the tracking error. In principle, DE could handle the problem, because the search space is a subspace of $[0\ldots1]^n$ and therefore all solution components are continuous numbers. However, despite that DE is a very effective algorithm for continuous, numerical problems, in our preliminary experimentation, we found that it fails to tackle the index tracking problem sufficiently well when used without any modifications. Instead, we implemented an additional operator that we call "position swapping", which swaps positions by exchanging the asset allocation weights between two assets with a zero and a non-zero allocation with a certain probability. With the extension of this operator the quality of the results and the required runtime could be substantially improved. This is perhaps not so surprising considering that the choice to use either an asset j or k requires that w_j is zero and w_k is redistributed to other weights or vice versa.

3.3 Constraint handling

As described in section 2, the index tracking problem is a minimization problem subject to constraints. In contrast to linear and quadratic programming, which can take advantage of information about constraints to find the optimal solutions[4], constraint handling is not part of search heuristics and need to be implemented separately.
Many approaches with different pros and cons have been suggested in the literature, such as penalty techniques, feasibility conserving operators, and repair methods (for an overview see Michalewicz and Fogel 2000).
The main idea of penalty techniques is to restate the problem by adding a penalty term $P(\mathbf{x}) \geq 0$, which accounts for constraint violations, to the objective function, i.e., $f'(\mathbf{x}) = f(\mathbf{x}) + P(\mathbf{x})$ for minimization problems. For a feasible solution, the penalty term $P(\mathbf{x})$ is zero.

[4] The global optimum of a linear and quadratic programming problem is always at a constraint border.

The penalty term can be defined in various ways, such as by the number of constraint violations, the amount of violations, or penalties that increase with the runtime. The main drawback of the penalty approach is its high parameter sensitivity, such as the choice of an appropriate penalty amount (Deb et al. 2002).

In case of index tracking, some constraints (such as (1) and (3)) can be tackled by "constructing" solutions that are guaranteed to be feasible (feasibility conserving operators) and by "repairing" infeasible solutions, i.e., by finding the nearest feasible solution to an infeasible solution. The remaining constraints can be tackled by a simple but effective technique related to the penalty approach without parameters, which is inspired by the constraint handling used in the NSGA-II and DEMPO algorithms (Deb et al. 2002, Krink and Paterlini 2007) for multiobjective optimization. The following pseudo-code illustrates the process of selecting the better among two solutions with the NSGA-II inspired approach:

Figure 1: Pseudo-code of the penalty approach related constraint handling in DECS-IT

```
if both solutions are feasible
     select the one with better fitness
end
if one solution is feasible and the other infeasible
     select the feasible solution
end
if both solutions are infeasible
     if one solution violates fewer constraints than the other
            select the solution that violates fewer
      else
            if one solution violates less much constraints than the other
                  select the solution that violates less much the constraints
            else
                  select the one with better fitness
            end
     end
end
```

3.4 Initialization

An often overlooked aspect of optimization with heuristics is the initialization of the process. A common default is to use random initialization, i.e., to generate a population of candidate solutions which have uniformly distributed random values for each of their vector components. However, when additional information about the problem at hand is available, a knowledge-based initialization can speed-up the search and improve the accuracy of the results. Our algorithm for

benchmark replication is no exception. Here, the main focus is on starting the search from a meaningful initial selection of asset positions.

Perhaps the most obvious way is to avoid positions that are highly correlated, because they can be expressed by one another. Instead one should rather choose a set of least correlated asset positions to have the maximum flexibility for replicating all aspects of the assets that are included in the benchmark. For this one would first cluster the assets based on their correlations and then pick representatives of each cluster. For instance (Focardi and Fabozzi, 2004), suggest that positions should be strictly chosen in accordance with this approach.

However, there are many parameters to set that can influence the quality of the results, such as the choice of the clustering method, the distance measure, the number of groups, and which assets to choose from each group. We decided to cluster the log-returns of the time series with hierarchical clustering using 'Ward' linkage and using the upper bound for the number of asset positions (K) as the number of groups to obtain the final data partition. From each group we select the asset position with the largest index weight within the group.

Another alternative to random initialization is to select the initial positions according to their weight in the benchmark, i.e., to choose the K positions that have the top-K largest weights in the index. The idea is that the best way to replicate an important asset is to use the asset itself. This can be easily implemented if the index composition is known at the beginning of the period. If such information is not available, market values or other values could be used as proxies, depending if the index is value-weighted or not. The idea is then sorting the index positions by their weights and to use the first K asset positions. In particular positions with very large asset weights should be included in the replication, whereas it makes less sense to include positions that are at the end of the top-K largest weights, where consecutive positions typically have very similar weights. For instance, considering the Nikkei 225 index for a replication with 30 assets, the index weights computed from market values at 12/01/2005 are equal to $w_{30} = 0.00743$ for the asset that ranks 30 in the top-n largest weights ranking and $w_{31} = 0.00737$ for the asset with rank 31. Hence, the preference for the former position is almost arbitrary.

In our algorithm, we implemented and compared the three initialization methods, which we will call henceforth *random*, *top-K least-correlated* and *top-K largest weights*. Empirical results are reported in section 4.3.3.

4 Experimentation and Results

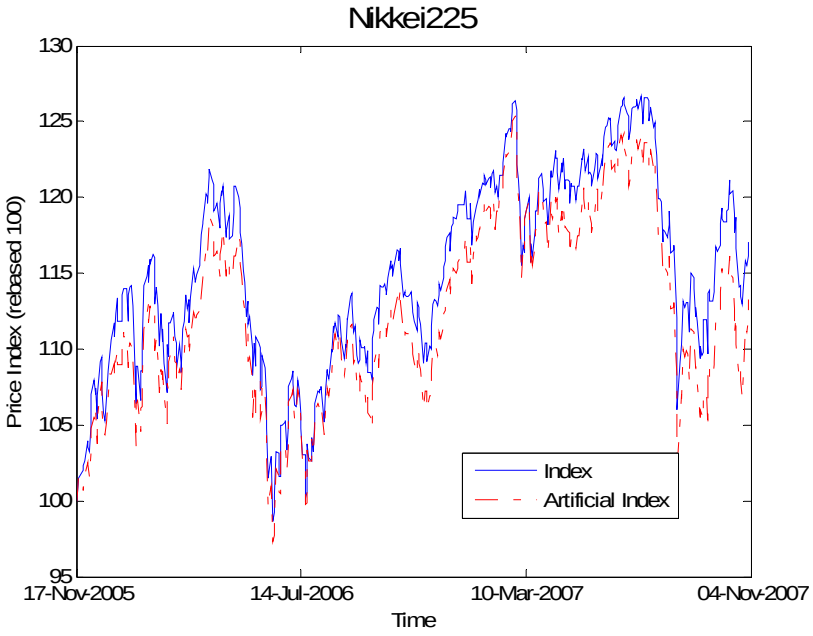
In the following sections we describe the runtime performance and our experiments on comparing the algorithm with QP on test cases. Finally, we discuss the results related to different seed initialization schemes. The reader is referred to Appendix A for a description of our preliminary experiments on parameter tuning, while Table 1 below report the DE parameter setting we used in our experimentations.

Table 1: Parameters of Differential Evolution (DE) and Quadratic Programming (QP). "Param." refers to the parameters, "Pop. size" to the population size and "Num.It." to the number of iterations. For DE: *CF*: crossover factor; *f*: scaling factor. For QP: *H* : Hessian matrix, i.e., second derivative search direction; F : first derivative vector, here zero vector because no linear components were defined in this version of the problem definition

DE		**DE** (weight fine-tuning and QP comparison)		**QP**	
Parameter	Value	Parameter	Value	Parameter	Value
Pop.size	50	Pop.size	200	*H*	cov. matrix Σ
Num.It.	20.000	Num.It.	5.000	*F*	zero vector
CR	0.7	*CR*	0.8		
f	0.3	*f*	0.3		

In the experimentation phase, we consider the Nikkei 225 price index and the stock prices of the 225 constituents from 17/11/2005-10/01/2007. We also build an artificial index, by using as index weights the market values (MV) observed on 01/12/2005, such that $wb_i = MV_i / \sum_{i=1}^{n} MV_i$ (see Figure 2) and normalize asset prices to 100 in order to compare DECS-IT with quadratic programming (QP).

Figure 2: Nikkei 225 index (rebased 100) in blue solid line, Nikkei 225 Artificial benchmark in red dashdot line.



4.1 Runtime performance

The DE search heuristic for index tracking has been implemented in MatLab 7.0. All experiments were done on a SONY VAIO PC laptop with 2.0 GHz Mobile Intel Pentium M processor, 798MHz, 1.00GB of RAM with Windows XP. Reasonable results, when tackling the index tracking problem without any simplifications, can be obtained in about 10 minutes (after 10.000 iterations) and high precision results within 20 minutes (after 20.000 iterations).

4.2 Preliminary experimentation and Test cases

We tested the functionality of our algorithm for a realistic index tracking problem, in which the benchmark portfolio consisted of 225 asset positions.
We ran three sets of experiments to investigate:
(i) the asset weight fine-tuning ability of DECS-IT,
(ii) the performance of DECS-IT compared to QP, and
(iii) the effect of different seed initializations on the optimization performance.

For the first and the second set of experiments we used simplified versions of the original index tracking problem by considering as objective function the one reported in (2) and excluding constraints (4-6). For experiment (i), we wanted to investigate the weight fine-tuning ability of our

algorithm independently of the asset position selection, whereas for the comparison with QP, we had to simplify the original problem to a quadratic programming problem.
The parameter settings for the optimization algorithms are shown in table 1 above.

4.2.1 Weight fine-tuning ability

In the first set of experiments, we tested our algorithm for its basic ability of numerical fine-tuning of the weights. For such task, we simplified the problem, such that there are no constraints on the number of asset positions. The optimal solution is obviously **w** = **wb**[5]. However, for a search heuristic starting with a population of random candidate solutions, the algorithm has to fine-tune all 225 weights precisely to obtain the correct result[6].
Table 2 shows the results for 30 runs, starting from random seed initialization (see section 3.4). It is clearly evident that DECS-IT was capable of approximating the correct result with high precision.

Table 2: Minimum, Mean, Maximum, Standard Deviation and 90% Percentile of best fitness values on 30 runs.

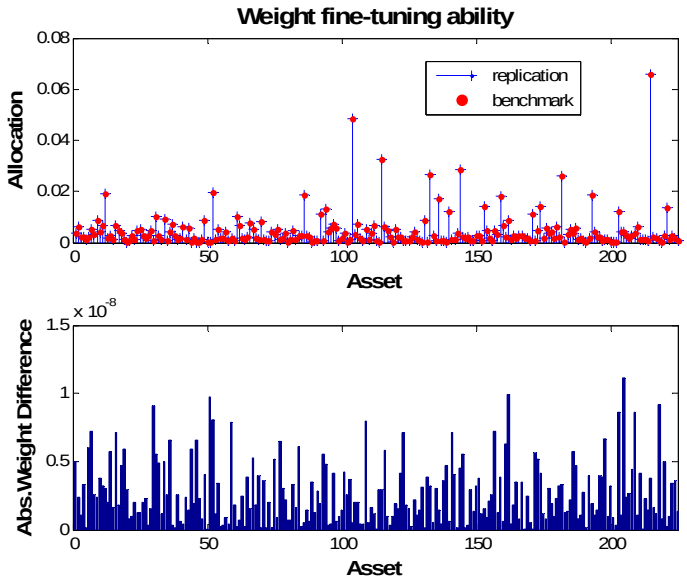
	Min	Mean	Max	Std	90%
Fitness (2)	2.23357E-19	2.23357E-19	2.23357E-19	9.8001E-35	2.23357E-19

Figure 3 shows a comparison between the actual index weights and the tracking portfolio (a) and the absolute value of the difference in the asset weights between the index weights and the tracking portfolio (b). The difference in asset position weights is negligible (with order of magnitude 10^{-8}).

[5] Index weights have been computed from market values keeping all the available decimals digits. In realistic instances, index weights would have a much smaller number of decimal digits.

[6] Note that in a realistic application, one would always use the benchmark itself (or a modified version if the benchmark violates any constraints) as one of the starting points for the search and thus the algorithm would not need to search at all.

Figure 3: Results of the asset weight-fine tuning experiment starting from random weights at initialization. Upper: Solution analysis plot showing the correspondence between the benchmark weights (red dots) and the replicated solution (black candle sticks) illustrated by allocation weights (y-axis) vs. asset positions (x-axis) for the final result. Lower: Absolute value of the difference of the index weight and the replication weight for each asset.



4.2.2 Comparison of DECS-IT with QP

As second test case, we wanted to compare the performance of the new algorithm to conventional optimization techniques, in particular to QP, since if the index weights are known, the objective function can be stated as in (2) and therefore it is quadratic and four out of six constraints are linear. As mentioned in section 1, QP cannot be used to tackle the type of index tracking problem that we consider in this paper. For a comparison with QP, we had to simplify the original problem to a quadratic programming problem. For this we neglected the task of finding the optimal subset selection of asset positions (constraint 4) and do not consider constraint 5. Then, we run two set of experiments:

i) we fix a priori the selection of K=50 positions (not optimized) and do not consider any lower/upper bound on the tracking portfolio weights (ε =0, ξ=1).

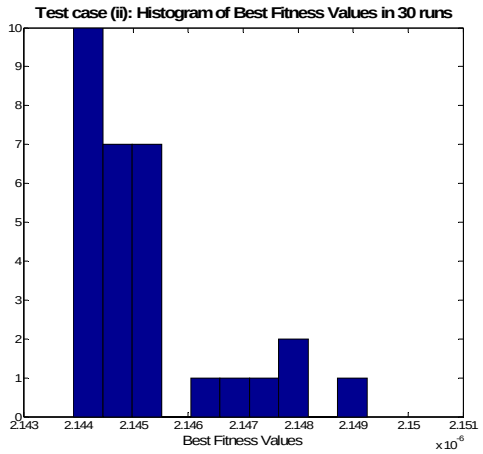
ii) we fix a priori the selection of K=100 positions (not optimized) and set ε=0, ξ=min(1,**wb**+0.01).

Table 3: QP and DECS-IT comparison on test cases. From column 2: QP result, Minimum, Mean, Maximum, Standard Deviation and 90% Percentile of best fitness values on 30 runs of DECS-IT algorithm.

	QP	DECS-IT				
	Fitness	Min	Mean	Max	Std	90%
Test case i	2.983E-06	2.983E-06	2.983E-06	2.983E-06	1.362E-21	2.983E-06
Test case ii	2.144E-06	2.144E-06	2.145E-06	2.149E-06	1.376E-09	2.148E-06

Table 3 shows that DECS-IT can obtain comparable results with QP. In the first set of experiments, DECS-IT obtains the exact same results of QP in all runs, while in the second set of experiments it obtains the exact same results of QP in 10 out of 30 cases (see Figure 4). In all the other cases, difference in the asset weights are negligible and would not have any real effect on a real-world application of the tracking portfolio. In fact, assets weights are not truncated to a given number of decimal places and differences have at most order of magnitude of 10^{-4}, which would hardly determine any difference in the number of asset shares to buy in the market.

Figure 4: QP and DECS-IT comparison on test cases. From column 2: QP result, Minimum, Mean, Maximum, Standard Deviation and 90% Percentile of best fitness values on 30 runs of DECS-IT algorithm.



4.2.3 DECS-IT performance for different seed initializations

After checking the weight fine-tuning capability of DECS-IT and comparing its performance with QP on test cases, we investigated the effect of different seed initialization. We set the constraints such that the 225 asset positions of the benchmark had to be replicated with a maximum of 50 of these positions, ε=0.00, ξ=0.1, Pop.size=50, Num.It=10000, *CR*=0.3, *f*=0.7. We considered the

NIKKEI 225 price index and the stock prices of the 225 constituents from 17/11/2005-01/11/2007. The market values used for computing the proxies for the index weights are observed on 01/12/2005.

In our experiments, we compared the performance of the three seed initialization schemes that we introduced in section 3.4. We considered then :
(a) *top-K largest weights* - asset positions with the top-50 largest weights.
(b) *top-K least-correlated* - asset positions of the top-50 least-correlated asset clusters,
(c) *random* - one random selection of 50 asset positions,

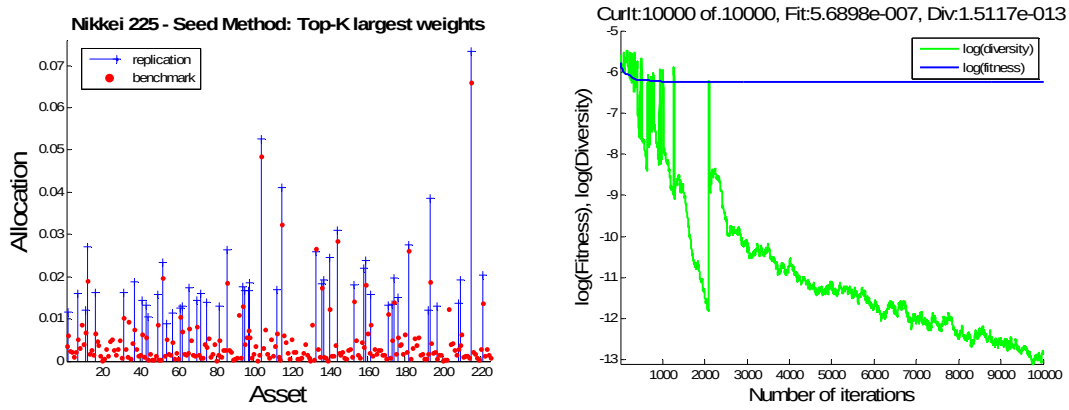
Figures 5(a), 5(b) and 5(c) show the results for single run of DECS-IT for different initialization scheme. The left plots show the tracking portfolio weights (blu candle sticks) compared to the corresponding index weights (red dots). The right plots show the logarithm of the best fitness value with respect to the number of iterations and the logarithm of the diversity across the DECS-IT population. Diversity has been defined as in Ursem (2002), which is the sum of the variance of the population divided by the size of the population and the diagonal of the search space[7].
As expected, DECS-IT obtained the best result $f(\mathbf{w})$ = 5.6898e-7 with the top-K largest weights initialization. The rights plots show that using the *top-K largest weights* converges faster to better fitness values than the other methods. However, premature convergence is avoided by using the position swapping operator that allows exploring new candidate solutions which are quite different, as the green spikes show, from the ones in the current population. The *top K-least correlated* initialization scheme shows a similar behavior with the *top-K largest weights*: the population diversity decreases fast at the beginning of the run, but the position swapping allows to explore new areas in the search space and to increase sharply again the population diversity. The *random* scheme does not exhibits such high spikes in the diversity of the population: the population diversity decreases slow at the beginning of the run since the initial population is widely scattered in the search space and promising solutions tend to compete with each other to attract the other individuals. Overall, the differences are rather small, although significant.
Finally, Figure 6 show the boxplots of the best reached fitness values in 30 runs for objective function (1) (left plot) and objective function (2) right plot for K=50, ε=0.00, ξ=0.1, Pop.size=50, Num.It=10000, CR=0.3, f=0.7. The *top-K largest weights* initialization seed scheme is to be preferred for both objective functions, while the *K-least correlated* method is not better than random initialization for objective function (1). To consider the K largest market values as starting

[7] (i.e.: div = sum(Variance(pop)) / (Pop.Size * L); L = sqrt(sum((ξ-ε).*(ξ-ε))))

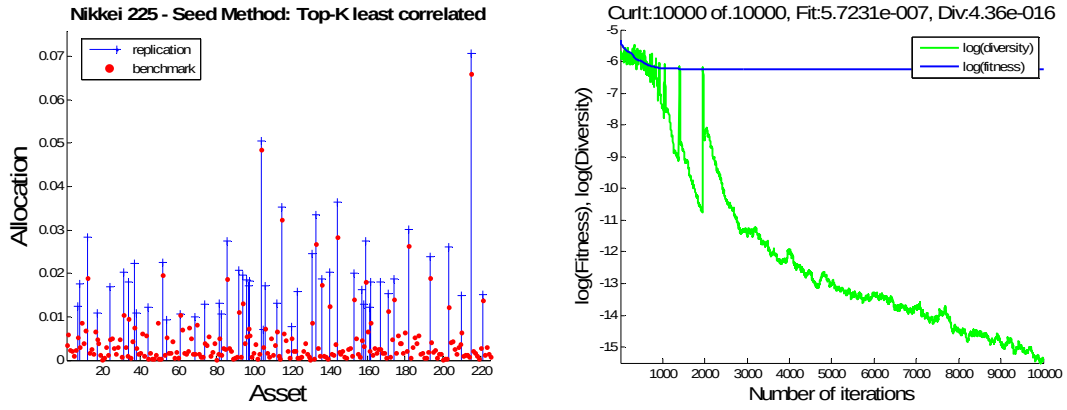
point for the tracking portfolio weights can be valuable even if **wb** is not explicitly considered in computing the value for function (1)), while combining such information with a selection mechanism based on clustering and choosing the *K-least correlated* positions (based on **Σ** estimates) does not seem to provide useful insight for the DE initialization.

Figure 5: Results of the DECS-IT experiments for the real index tracking problem using respectively (a) *top-K largest weights,* (b) *top-K least-correlated* and (c) random initializations schemes. Left: Solution analysis plot showing the correspondence between the benchmark weights (red dots) and the replicated solution (blue candle sticks) illustrated by allocation weights (y-axis) vs. asset positions (x-axis) for the final result. Right: log best fitness (blu) and log population diversity (green) over number of iterations.

(a) Results using top-*K* largest weights initialization



(b) Results using top-*K* least-correlated initialization



(c) Results using random initialization with one random selection of asset positions

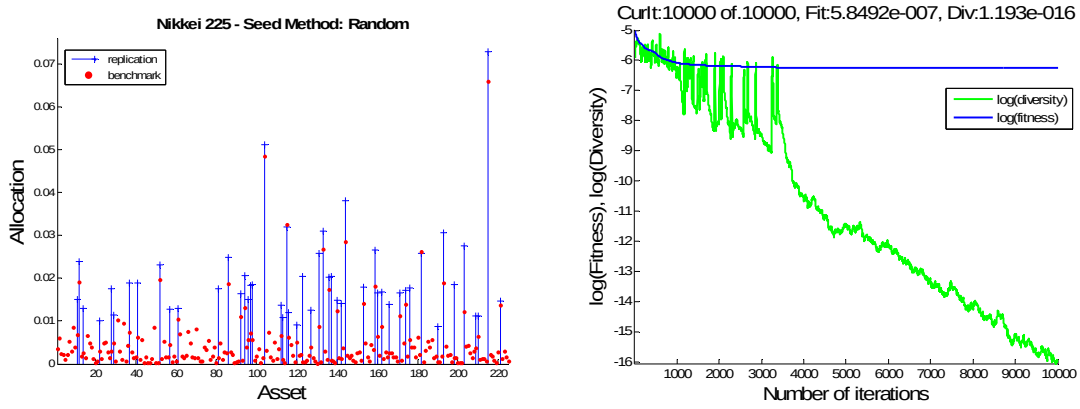
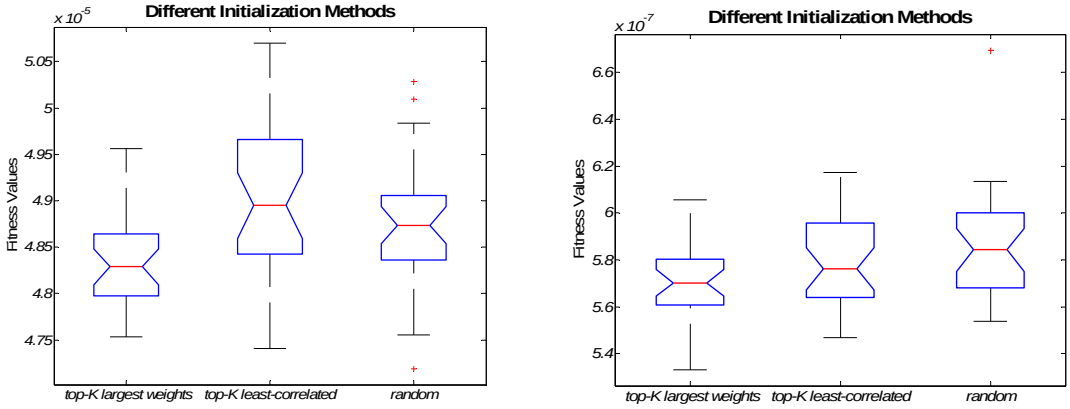


Figure 6: Boxplots of best fitness values in 30 runs for the real index tracking problem using respectively, (a) *top-k largest weights* (b) *top-K least-correlated* and (c) *random* initializations schemes. Left: Boxplots of the best fitness values for objective function (1), Right: Boxplots of the best fitness values for objective function (2)



5 Empirical Analysis

In this section, we discuss the empirical results related to a realistic application of the methodology over different time horizon. We investigate the in-sample and out-of-sample performances of the optimal tracking portfolios on Dow Jones 65 (Period: 13/04/02- 20/12/03) and Nikkei 225 (Period: 18/11/05-27/07/07) indexes. Table 4 and Figure 7 show the descriptive statistics and the behavior of the two indexes. We aim at determining the optimal index tracking portfolios by considering periodic rebalancing of the assets weights. We use a rolling-windows scheme: for each index, we determine the optimal tracking portfolio considering daily log-returns on a window [*T, T+200-1*]

and evaluate its out-of-sample performance on window [*T+200, T+200+20*] for *T* from 1 to 221 with step size 20. Hence, we consider 12 periods in which to rebalance our portfolios.

Our aim is to determine the optimal tracking portfolio with respect to the objective function (1), considering constraints (1-5) and setting the optimization parameters such that ε_i=0.01, ξ_i=0.1 (*i*=1,…,*n*), *Lb*=0.05, *Ub*=0.4, L=0. We investigate the effect of varying the maximum number of assets *K* that can be included in the index tracking portfolios. Since so far we have not considered transaction cost, we introduce the following constraint: $\sum_{i=1}^{n} |\Delta w_i| \leq 0.1$, that is the total change in asset weights from the allocation at the previous time-step must be lower than 0.1. This constraint allows controlling for transaction costs when updating the tracking portfolio from one period to the other. The DECS-IT algorithm can easily deal with such constraint, even if is non linear. If such constraint is not considered, the changes in asset weights among two consecutive periods would be extremely large[8], such that realistic the index tracking strategy over time would be too expensive. Transaction costs could be also explicitly considered as an optimization task and further investigation is high on our agenda.

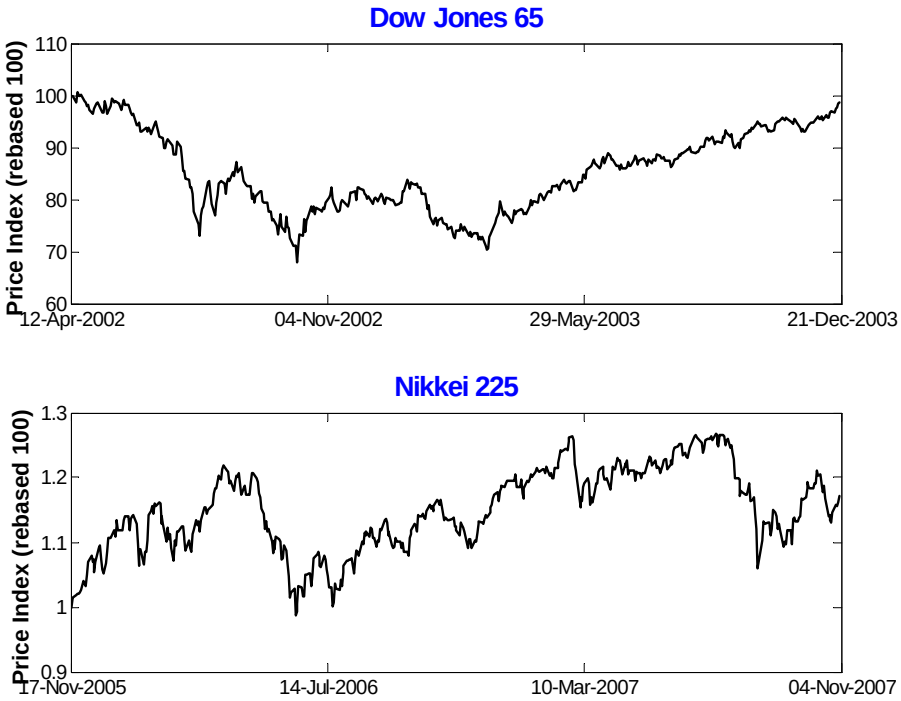
We set DECS-IT parameter values such that: Pop.size is 100, the number of iterations Num.It. is 10.000, the crossover rate *CR* is 0.7 and the scaling factor *f* is 0.3. We use the top-*K* largest weight seed initialization scheme, using proxies from market values for the index weights.

Table 4: Descriptive Statistics of Dow Jones 65 and Nikkei 225

	Sample Size	Mean	Std. Dev.	Skewness	Kurtosis	Min	Max
Dow Jones 65	440	0.0000	0.0134	0.1917	4.3976	-0.0461	0.0535
Nikkei 225	440	0.0005	0.0112	-0.2633	3.8453	-0.0423	0.0352

[8] Empirical results are available upon request.

Figure 7: Dow Jones 65 and Nikkei 225 indexes (rebased 100)



Tables 5 and 6 show the empirical results for the Dow Jones 65 and the Nikkei 225 indexes. In the first three columns, we report respectively the minimum, the average and the maximum number of assets included in the tracking portfolios. We consider K={20,25,30,35,40,45,50,55} for the Dow Jones 65 and K={20,30,40,50,60,70,80,90} for the Nikkei 225.

For each index, we then compute the following statistics:

- In-Sample[9] and Out-of-Sample Annualized Tracking Error Volatility (columns 4-5)
- In-Sample and Out-of-Sample Annualized Excess Return (columns 6-7)
- Out-of-Sample Information Ratio (column 8)
- Average Turnover, calculated as $Av.Turnover = \sum_{i=1}^{n} |\Delta w_i| / 2$ (column 9)
- Out-of-Sample Correlation w.r.t. the Index (column 10)
- Out-of-Sample Beta w.r.t. the Index (column 11)

[9] The in-sample annualized tracking error volatility corresponds to the average of the annualized in-sample tracking error volatility computed in the 12 different time periods.

As we expected, if we increase the maximum number of assets that can be included in the tracking portfolio, the in-sample annualized tracking error volatility tend to decrease. However, for K=55 for Dow Jones 65 the in-sample tracking error volatility is larger than for K=50. This is not due to the convergence of a suboptimal solution by the DECST-IT algorithm but because of the presence of constraints in the optimization problem. In fact, the in-sample TEV is smaller in the first 200 observations window, but the constraints on the turnover and the asset weights minimum and maximum bound lead to identify suboptimal portfolios in the other windows, compared to the case when K=50. If we considered the Nikkei 225, we notice that even if we fix K=90 (last row), the optimal portfolio includes at most 85 assets. Furthermore, it seems that the relationship between the maximum number of assets K and the out-of-sample tracking error volatility is also decreasingly monotonic (see column 5) but further investigation on different indexes should be considered. As a strategy to choose the maximum number K of assets to include in the tracking portfolio, we could then decide to stop the algorithm when the average in-sample tracking error volatility stops decreasing. However, such strategy could lead to choose a high number of K, which is often suboptimal when transaction and monitoring costs and other statistics are considered. In fact, we notice that there is not an increasingly monotonic relationship between the maximum number of assets K and the in-sample and out-of-sample annualized excess returns: the maximum in-sample and out-of-sample excess returns are respectively for K=30 and K=25 for Dow Jones 65 and for K=20 for Nikkei 225. The out-of-sample information ratio, which as been computed as the ratio between the out-of-sample excess return and the tracking error volatility, measures the excess return of an investment manager divided by the amount of risk the manager takes relative to an index. Such ratio allows considering jointly risk and return of the tracking portfolio: from columns 8, we notice that it has maximum value for K=25 for Dow Jones and for K=20 for Nikkei 225, suggesting that a tracking portfolio with a limited number of assets should be preferred. Column 9 shows that the constraint on the total turnover is always satisfied and we could than estimate the transaction costs as proportional to the tracking portfolio turnover. Finally, columns 10 and 11 report the out-of-sample correlations with respect to the index and the betas with respect to the index. Correlation or beta equal 1 are highly appealing. Increasing K does not necessarily lead to higher out-of-sample correlation or beta closer to 1: we have the best performance with respect to correlation and beta for K=45 for Dow Jones 65 and respectively K=70 and K=30 for Nikkei 225.

Summing up, choosing the maximum number of assets to include in the tracking portfolio is a difficult task: a large number of assets, a part from being often more expensive in term of transaction and monitoring, could lead to tracking portfolios with lower tracking error volatility but smaller excess return, information ratio, correlation and beta. On the other hand, a more

parsimonious choice of K, could allow not enough diversification identifying tracking portfolios with risk-profile different from the index. Further investigation is currently under development.

Table 5: In-sample and Out-of Sample results for Dow Jones 65

Dow Jones 65										
Number of Assets included in the Tracking Portfolio			**Annualized Tracking Error Volatility**		**Annualized Excess Return**		**Information ratio**	**Average Turnover**	**Correlation**	**Beta**
Min	*Mean*	*Max*	*In-Sample (average)*	*Out-of-Sample*	*In-Sample*	*Out-of-Sample*	*Out-of-Sample*	*Out-of-Sample*	*Out-of-Sample*	*Out-of-Sample*
20	20.00	20	0.214%	3.64%	0.23%	-6.80%	-1.87	3.67%	99.17%	0.72
25	25.00	25	0.1748%	2.97%	1.94%	3.87%	1.30	4.11%	99.81%	1.10
30	30.00	30	0.1441%	2.52%	2.21%	-0.39%	-0.15	4.57%	99.80%	0.94
35	35.00	35	0.1319%	2.38%	1.50%	-0.78%	-0.33	4.38%	99.82%	0.94
40	40.00	40	0.1181%	2.18%	1.79%	-0.23%	-0.10	4.58%	99.86%	0.95
45	45.00	45	0.1027%	1.89%	1.83%	0.08%	0.04	4.52%	99.94%	0.98
49	49.92	50	0.0928%	1.70%	2.08%	1.20%	0.70	4.29%	99.94%	1.04
52	53.58	55	0.0932%	1.64%	2.09%	1.49%	0.91	4.24%	99.93%	1.04

Table 6: In-sample and Out-of Sample results for Nikkei 225

Nikkei 225										
Number of Assets included in the Tracking Portfolio			**Annualized Tracking Error Volatility**		**Annualized Excess Return**		**Information ratio**	**Average Turnover**	**Correlation**	**Beta**
Min	*Mean*	*Max*	*In-Sample (average)*	*Out-of-Sample*	*In-Sample*	*Out-of-Sample*	*Out-of-Sample*	*Out-of-Sample*	*Out-of-Sample*	*Out-of-Sample*
20	20.00	20	0.210%	3.53%	2.19%	1.48%	0.42	4.11%	99.27%	1.10
30	30.00	30	0.155%	2.81%	0.71%	-0.74%	-0.26	4.53%	98.75%	0.97
40	40.00	40	0.127%	2.06%	0.14%	-2.43%	-1.18	4.30%	99.57%	0.86
50	50.00	50	0.115%	1.80%	-0.38%	-2.01%	-1.12	4.57%	99.14%	0.87
60	60.00	60	0.105%	1.68%	0.16%	-2.52%	-1.50	4.58%	98.29%	0.79
69	69.50	70	0.100%	1.40%	0.77%	-2.40%	-1.72	4.56%	99.40%	0.89
75	77.75	80	0.098%	1.46%	0.27%	-4.55%	-3.11	4.48%	97.46%	0.70
74	78.42	85	0.097%	1.43%	-0.14%	-2.40%	-1.68	4.39%	98.56%	0.80

6 Discussion and Conclusions

A quantitative approach to index tracking is a challenging econometric and optimization problem. In this work, we focus on the optimization problem, proposing a new heuristics that can successfully tackle it in reasonable runtime. The optimization requires selecting a subset of optimal asset positions from the benchmark while finding the optimal asset allocation weights and respecting additional linear and non-linear constraints. Moreover, the problem is high-dimensional. Although the objective function of this problem could often be stated as quadratic it is not a quadratic programming problem, because of non-linear constraints, and thus cannot be tackled by quadratic programming (QP). Our solution has been to develop a search heuristic (DECS-IT) that combines the excellent performance of Differential Evolution (DE) for continuous, numerical optimization with a combinatorial search operator, seed initialization and a particular selection of constraint handling techniques. Our initial implementation of DECS-IT without an additional operator to search for the optimal asset positions turned out to be incapable of solving this problem sufficiently well. However, we have then shown that a modified version of Differential Evolution with some simple operators can also tackle the index tracking continuous numerical problem remarkably well.

Our final experiments showed that a good starting point for the search is simply to select the asset positions that have the largest weights in the benchmark, even if random initialization can also lead to meaningful results. This makes particular sense for the very largest weights and becomes less meaningful for smaller weights especially at the cut-off between asset positions that should be included. The latter typically have very similar weights, which means that the selection of positions is rather arbitrary. This simple approach turned out to be slightly better than to use asset positions that are least correlated and the random selection of positions. Regarding the use of least correlated positions, we would like to point out that we only investigated one reasonable out of many possible settings. Perhaps a combination of selecting the largest and the least correlated positions would work fine if assets with very similar weights are selected according to their correlation. We did not investigate this further at this stage, since the results from selecting assets with the largest weights initialization were already fully satisfying, even when using market values as proxies for the index weights.

Another interesting outcome is that the DECS-IT algorithm can obtain the same quality of results as QP for simplified quadratic programming problem instances of the original index tracking problem. As expected, the computation time for DECS-IT is much longer than for QP (6-7 minutes compared to 7 seconds on PC laptop with 2.0 GHz Intel Pentium M), but this is a small price to pay considering the advantage of being able to tackle an index tracking problem in full generality.

In our investigation, we test DECS-IT on two different objective functions. DECS-IT could easily deal with more complex objective functions and other non-linear constraints as for example the one on total turnover, which we have introduced in section 5.

Finally, we have tested DECS-IT when considering the index-tracking problem on different time periods for two equity indexes: the Dow-Jones 65 and Nikkei 225. We have shown that increasing the maximum number of assets to be included in the tracking portfolios decreases the in-sample and out-of sample tracking error volatility up to a certain limit (K=50 out of 65 for Dow Jones, and K=85 out of 225 for Nikkei 225), which strongly depend on the index composition[10]. However, increasing K, a part from being often more expensive in term of transaction and monitoring costs, does not necessarily lead to tracking portfolios with better annualized excess return, information ratio, correlation and beta.

Further research on different indexes and with different portfolio optimization setting is currently high on our agenda, as well as, studying how different covariance estimates can affect the index tracking problem.

References

Beasley, J.E, Meade, N, Chang, T.J. (2003). An evolutionary heuristic for the index tracking problem. European Journal of Operational Research, 148, 621–643.

Chang, T.J., Meade, N., Beasley, J.E., Sharaiha, Y.M. (2000). Heuristics for Cardinality Constrained Portfolio Optimization, Computers and Operations Research, 27, 1271-1302.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm, IEEE Transactions on Evolutionary Computation, 6, 2, 182-197.

Dueck, G., & Winker, P. (1992). New concepts and algorithms for portfolio choice, Applied Stochastic Models and Data Analysis, 8, 159-178.

Focardi, S.M., & Fabozzi, F.J. (2004). A methodology for index tracking based on times-series clustering, Quantitative Finance, 4, 417-425.

Fogel, L.J., Owens, A.J., Walsh, M.J. (1966). Artificial intelligence through simulated evolution, New York, John Wiley.

Gilli, M. & Këllezi, E. (2002). The threshold accepting heuristic for index tracking. In Financial Engineering, E-Commerce, and Supply Chain, Pardalos P, Tsitsiringos V (eds). Kluwer Academic Press: Boston, MA.

[10] Nikkei 225 has a smaller number of asset weights larger than 5% and 1% than Dow Jones 65.

Holland. J.H. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Harbor.

Kennedy, J. & Eberhart, R.C.(1995). Particle swarm optimisation. In: Proc. of the 1995 IEEE International Conference on Neural Networks, IEEE Press, Piscataway, NJ, 4, 1942-1948.

Kirkpatrick,S., Gelatt, C.D. & M.P. Vecchi (1983) Optimization by Simulated Annealing Science, 220, 671-680.

Kwiatowski, J.W. (1992). Algorithm for index tracking. Journal of Mathematics Applied in Business and Industry, 4, 279–299.

Krink, T., Filipi˘c, B., Fogel, G., Thomsen, R. (2004). Noisy optimisation problems—a particular challenge for differential evolution? In: Proceedings of the Sixth Congress on Evolutionary Computation (CEC-2004). IEEE Press, Piscataway, NJ, USA, 332–339.

Krink T., Paterlini S., Resti A. (2007). Using Differential Evolution to improve the accuracy of bank rating systems. Computational Statistics & Data Analysis, Elsevier, 52/1, 68-87.

Krink T., Paterlini S., Resti A. (in press).The optimal structure of PD buckets, Journal of Banking and Finance.

Krink T., Paterlini S. (2007). DEMPO- Differential Evolution for Multiobjective Portfolio Optimization, (http://ideas.repec.org/p/mod/wcefin/08012.html)

Lampinen, J., A Bibliography of Differential Evolution Algorithm, http://www2.lut.fi/~jlampine/debiblio.htm

Maringer, D., & Oyewumi, O. (2007). Index Tracking with Constrained Portfolios. Intelligent Systems in Finance, Accounting and Management, 15, 57–71.

Maringer, D., & Parpas, P. (in press) Global Optimization of Higher Order Moments in Portfolio Selection, Journal of Global Optimization.

Michalewicz, Z., & Fogel, D.B., (2004) How to solve it: modern heuristics, Springer.

Paterlini S., Krink T., (2006). Differential Evolution and Particle Swarm Optimisation in Partitional Clustering. Computational Statistics & Data Analysis, Elsevier, 50/5, 1220-1247.

Pope, P. & Yadav, P.K., (1994). Discovering error in tracking error. Journal of Portfolio Management 20, 27–32.

Price, K., Storn, R., Lampinen, J., (2005). Differential Evolution: A Practical Approach to Global Optimization. Springer.

Rechenberg, I., (1973). Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog, Stuttgart.

Rudolf, M., Wolter, H., Zimmerman, H., (1999). A linear model for tracking error minimization. Journal of Banking and Finance, 23, 85–103.

Storn, R., & Price, K., (1997). Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11,4, 341–359.

Ursem, R. K., (2002). Diversity-guided evolutionary algorithms. In Proc. 7th Int. Conf. Parallel Problem Solving From Nature, 2439, LNCS, J. J. M. Guervs et al., Eds., Granada, Spain.

Vesterstrøm J., & Thomsen R., (2004). A comparative study of Differential evolution, Particle Swarm Optimisation and Evolutionary Algorithms on Numerical benchmark problems. In: Proc. of the Sixth Congress on Evolutionary Computation (CEC-2004). IEE Press, Piscataway, NJ, USA, 1980-1987.

Appendix A – Parameter Tuning

The performance of search heuristics, i.e., their accuracy and runtime, depends on the problem at hand and the choice of the algorithmic parameters. The process of finding the optimal parameters of a search heuristic is usually referred to as tuning. Compared to other heuristics, such as GA, DE does not have many parameters and requires very little tuning, as it has been already shown in other studies (Maringer and Parpas forthcoming, Paterlini and Krink, 2006, Krink and Paterlini 2007).
In our parameter tuning experimentation, we consider the problem of minimizing the tracking error volatility as defined in (1) subjects to constraints (1 - 4) with ε=0.01, ξ=0.10[11], K=50. To test the algorithm we consider the NIKKEI 225 price index and the stock prices of the 225 constituents from 17/11/2005-10/01/2007.
Using Differential Evolution requires setting the values of only few parameters: the population size (Pop.size), the number of iterations (Num.It), the crossover rate (*CR*) and the scaling factor (*f*). In our preliminary experimentations we consider 30 runs for each combination of the following parameters setting: Pop.size: {50, 100}, Num.It.:{2500, 5000, 10000}, *CR*:{0.7, 0.8, 0.9} and *f*:{0.2 0.3 0.4}. Hence, empirical results on parameter tuning are based on 1620 runs of DECS-IT algorithm. Figure A.1- A.4 show the main results of our investigation.

In realistic applications, such as index tracking, the performance bottle-neck of search heuristics is the objective function evaluation. The number of evaluations is equal to the population size (number of candidate solutions that are refined in each iteration) times the number of iterations of optimization heuristic plus the comparably small number of evaluations during the initialization. Figure A.1 (a) shows that the larger the number of iterations the less dispersed the distribution of the best fitness values and the more effective the convergence to smaller (better) fitness values. Furthermore, Figure A.1 (b) suggests that larger population's sizes yield better results if the number of evaluations is large enough (i.e.: 5000, 1000).

The scaling factor is usually chosen in the interval [0.2-0.4]. Figure A.2 (a) shows that there is no great difference in the best recorded fitness values distributions for scaling factor equal to 0.3 and 0.4, whilea scaling factor equal to 0.2 leads to slower convergence. Figure A.2 (b) points out that the choice of the scaling factor tends to affect less the algorithm convergence as the number of

[11] Please, notice that setting ε=0.01 implies that at most 100 assets would be included in the tracking portfolio and setting ξ=0.10 implies that at least 10 assets must be included.

iterations increases. Such result is not surprising since the scaling factor decreases the vector difference between two DE individuals before adding it to a third one. Hence, it affects the length of the step size in exploring the search space: a small scaling factor does not allow fast exploration on the whole search space. Therefore, a higher number of iterations are usually required to converge to better results.

The crossover rate usually varies in the interval [0.7-0.9]. Figure A.3 (a) shows that choosing a crossover rate equal to 0.7, 0.8 or 0.9 does not influence the algorithm convergence, since the three boxplots have similar distributions for the best fitness values. Furthermore, Figure A.3 (b) points out, as we expected, that the crucial parameter is the number of iterations. In fact, increasing the number of evaluations from 2500 to 10000 leads to converge steadily towards lower (best) fitness values and less spread distributions no matter which crossover rate we choose.

Finally, as figure A.4 shows, the choice of a scaling factor equal to 0.2 seems to slow down the algorithm convergence no matter which crossover rate we choose. Results suggest that a scaling factor equal to 0.2 could be too low for even 10000 iterations, while all other parameters do not lead to remarkable difference on the quality of the results. However, the reader should notice that the order of magnitude of the difference in the best reached fitness is very small. The reader is refereed to Krink and Paterlini (DEMPO-2007) and to Krink et. al (2007) for further evidence about the robustness of DE with respect to parameter choices.

Figure A.1: Boxplots of best recorded fitness values for number of evaluations {2500, 5000, 10000} for 30 runs for each combination of the following parameter settings: population size {50, 100}, crossover rate {0.7, 0.8, 0.9} and scaling factor {0.2, 0.3, 0.4}; (a) Boxplots of best fitness values for number of evaluations {2500, 5000, 10000} and population size {50, 100} for 30 runs for each combination of the following parameter settings: crossover rate {0.7, 0.8, 0.9} and scaling factor {0.2, 0.3, 0.4};

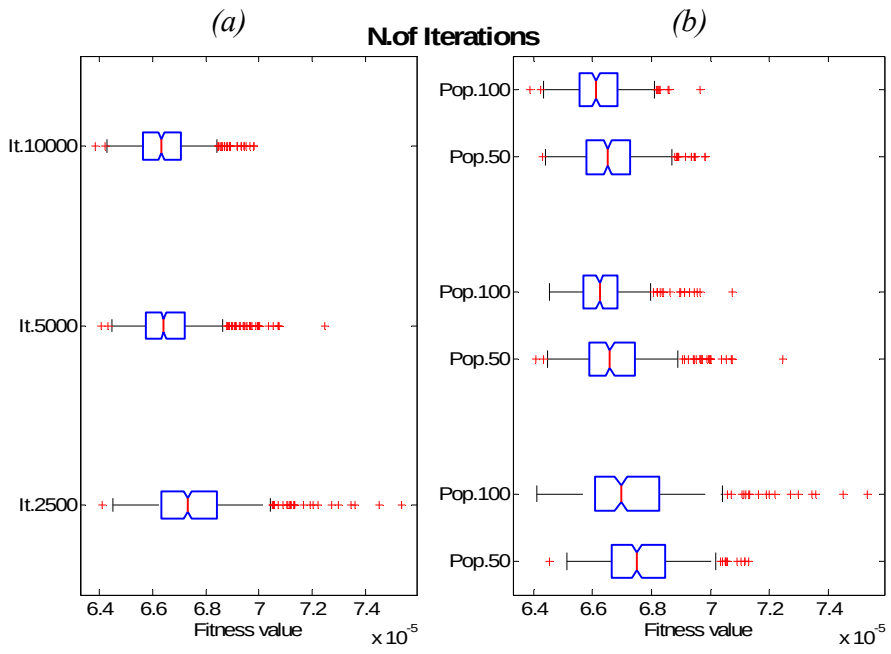


Figure A.2: (a) Boxplots of best fitness values for scaling factor {0.2, 0.3, 0.4} for 30 runs for each combination of the following parameter settings: number of iterations {2500, 5000, 10000},population size {50, 100}, crossover rate {0.7, 0.8, 0.9}; (b) Boxplots of best fitness values for scaling factor {0.2, 0.3, 0.4} and number of evaluations {2500, 5000, 10000} for 30 runs for each combination of the following parameter settings: number of iterations {2500, 5000, 10000},population size {50, 100}, crossover rate {0.7, 0.8, 0.9}.

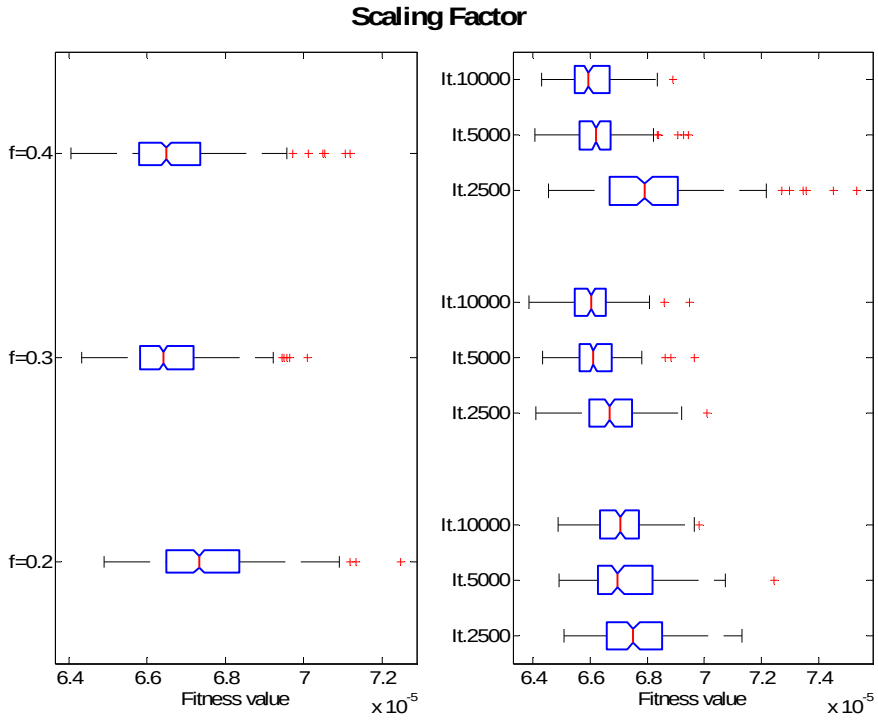


Figure A3: (a) Boxplots of best fitness values for crossover rate {0.7, 0.8, 0.9} for 30 runs for each combination of the following parameter settings: number of iterations {2500, 5000, 10000},population size {50, 100}, scaling factor {0.2, 0.3, 0.4}; (b) Boxplots of best fitness values for crossover rate {0.7, 0.8, 0.9} and number of evaluations {2500, 5000, 10000} for 30 runs for each combination of the following parameter settings: number of iterations {2500, 5000, 10000},population size {50, 100}, scaling factor {0.2, 0.3, 0.4}.

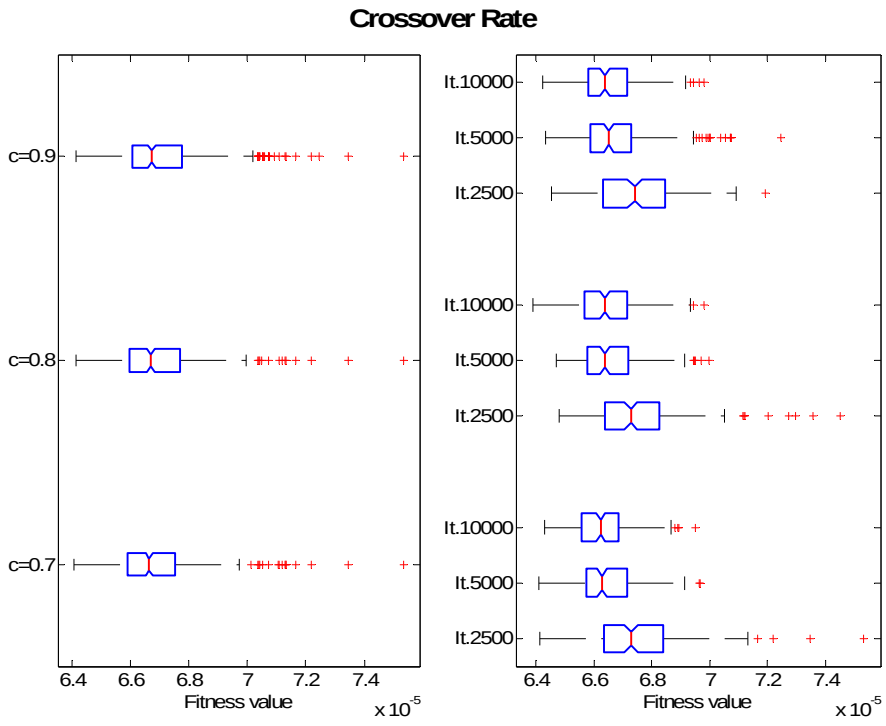
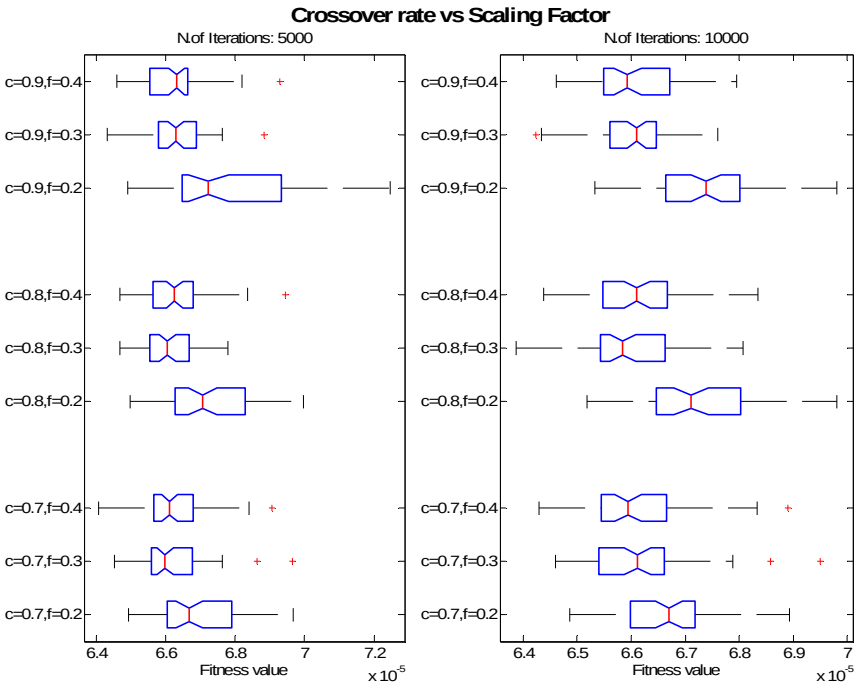


Figure A.4: (a) Boxplots of best fitness values in 5000 iterations for crossover rate {0.7, 0.8, 0.9} and scaling factor {0.2, 0.3, 0.4} for 30 runs for each combination of the following parameter settings: population size {50, 100}, crossover rate {0.7, 0.8, 0.9}, scaling factor {0.2, 0.3, 0.4}; (b) Boxplots of best fitness values in 10000 iterations for crossover rate {0.7, 0.8, 0.9} and scaling factor {0.2, 0.3, 0.4} for 30 runs for each combination of the following parameter settings: population size {50, 100}, crossover rate {0.7, 0.8, 0.9}, scaling factor {0.2, 0.3, 0.4}.



Appendix B – Pseudo-code

Figure B.1: Pseudo-code of Rand/1/Exp Differential Evolution.

```
function Differential_Evolution()

      initialize();
      evaluate();
      while curit<NumIt
            for i=1:popSize
                  MutateandRecombine();
                  Evaluate()
                  if fitness(offspring)>fitness(parent),
                        choose offspring;
                  else
                        choose parent;
                  end
            end
      end
function MutateandRecombine(){
```

$$c.\text{gene}_j = \begin{cases} m.\text{gene}_j + f\cdot(k.\text{gene}_j - l.\text{gene}_j) & \text{if } U(0,1) < cf \\ i.\text{gene}_j & \text{otherwise} \end{cases}$$

Figure B.2: Pseudo-code of DECS-IT algorithm

```
function DECS-IT()

% Initialize and evaluate the start population of candidate solutions
GenRandomPop(); % Initialize population
for i=1:popSize  EvaluateFitnessFunction(); end

Record and save results
curIt=1;

% Iteratively improve the population of candidate solutions
while curIt<numIt
  for i=1:popSize

% Apply DE with operator "Rand\1\Exp"
%select three other candidates pop(i1), pop(i2), and pop(i3) randomly
cand = pop(i3,:) + f * (pop(i1,:)-pop(i2,:));
cr_numDimCopy = round(numParam*(1.0-cr));
for j=1:cr_numDimCopy
  nsel = ceil(numParam*rand());
  cand(nsel) = pop(i,nsel);
end

% Rescale (such that SUM(cand)=1.0) and repair candidate cand
cand = cand / sum(cand);
RepairBoundsFeasibility();

% Apply operator to explore the asset position selection
cand = SwapZeroPositions(cand);

% Evaluate the new candidate solution cand
EvaluateFitnessFunction(cand);

% Select the new candidate if it is better than pop(i)
```

```
if(cand and pop(i) are feasible and cand_fitness<fitness(i))
   OR (cand is feasible and pop(i) is infeasible)
   OR (cand violates less many constraints than pop(i))
   OR (cand and pop(i) violate the same number of constraints,
   but cand with less amount)
   OR (cand and pop(i) violate the same number and with the same
   amount and cand_fitness<fitness(i))

  pop(i) = cand;
end
  end % for i=1:popSize

  Record and save results
  curIt = curIt + 1;

end % while curIt<numIt
```