www.cefin.unimore.it

CEFIN Working Papers
No 16

Differential Evolution and Combinatorial Search for Constrained Index Traking

by T. Krink, S. Mittnik, S.Paterlini

March 2009

CEFIN – Centro Studi di Banca e Finanza
Dipartimento di Economia Aziendale – Università di Modena e Reggio Emilia
Viale Jacopo Berengario 51, 41100 MODENA (Italy)
tel. 39-059.2056711 (Centralino) fax 059 205 6927

Differential Evolution and Combinatorial Search for Constrained Index Tracking

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

[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]*Center for Financial Studies, Goethe University, Grüneburg Platz 1, 60323 Frankfurt, Germany*

[d]*Dept. of Economics, CEFIN & RECent, University of Modena and Reggio E., Viale J. Berengario, 51,41100 Modena, Italy*

Abstract

Index tracking is a valuable low-cost alternative to active portfolio management. The implementation of a quantitative approach, however, is a major 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 subset of assets and to fine-tune their weights. The former is a combinatorial, the latter a continuous numerical problem. Both problems need to be addressed simultaneously, because whether or not a selection of assets is promising depends on the allocation weights and vice versa. Moreover, the problem is usually of high dimension. Typically, an optimal subset of 30-150 positions out of 100-600 need to be selected and their weights determined. Search heuristics can be a viable and valuable alternative to traditional methods, which often cannot deal with the problem. In this paper, we propose a new optimization method, which is partly based on Differential Evolution (DE) and on combinatorial search. The main advantage of our method is that it can tackle index tracking problem as complex as it is, generating accurate and robust results.

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

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 matrices, such that the tracking portfolio can replicate the index not only in sample but also out of sample. The reader is referred 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, this 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., the concentration limits of the UCITS rules[1], which are binding for most portfolios in the EU, requires that the sum of all asset weights exceeding 5% must be smaller than 40%).

An 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 quadratic objective function. Optimization heuristics can even be applied when the optimization problem

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

cannot be stated formally by a set of equations. The only prerequisite is that there is some way of comparing the quality of solutions, here the quality of asset weight vectors with respect to 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). These heuristics have already been successfully applied to the index tracking problem (see Beasley et al. 2003 and references therein). Recent studies, however, show that GAs and SA are poor choices when tackling non-trivial continuous numerical problems. Instead, Differential Evolution (DE) (Storn and Price 1997) and evolution strategies (ES) (Rechenberg 1973) should be preferred. DE has shown a remarkable performance in continuous numerical problems, compared to other heuristics, such as particle swarm optimization (Kennedy and Eberhart 1995) and genetic algorithms (Vesterstrøm and Thomsen 2004, Paterlini and Krink 2006; Krink et al. 2007). Research on index tracking (and on non-continuous numerical problems) using DE is still at an early stage, but recent results seem to support its usefulness in such applications, though further analysis is still required (Maringer and Oyewumi 2007). In fact, even though DE specializes on continuous, numerical optimization, we believe that it could be easily adapted to successfully tackle combinatorial problems, such as the index tracking.

In this work, we propose a hybrid algorithm – henceforth, Differential Evolution and Combinatorial Search for Index Tracking DECS-IT – that extends DE with an additional combinatorial search operator to determine the optimal choice of asset positions. 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 realistic setting, we consider common real-world constraints portfolio managers face and implement a constraint handling routine using a combination of techniques, which are specifically tailored to the index tracking problem. Our empirical results show that the proposed method can tackle complex index tracking problems while generating very accurate and robust results.

The remainder of this paper is organized as follows. In Section 2, we formally define the index tracking problem as an error minimization problem. Section 3 outlines the new optimization algorithm; and in Section 4 we present results on seed initialization and on a comparative study with quadratic programming regarding using 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 Problem

We state the optimization goal of the index tracking problem as a search for the portfolio that minimizes the tracking error volatility with respect to a given benchmark (index). Formally, we write:

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

subject to

(c1) $\sum_{i=1}^{n} w_i = 1$

(c2) $0 \le w_i \le 1$

(c3) $\varepsilon_{\mathrm{i}}\delta(w_i) \le w_i \le \xi_i\delta(w_i),\ \delta(w_i) = \begin{cases} 1 & \text{if } w_i > 0 \\ 0 & \text{otherwise} \end{cases}$

(c4) $L \le \sum_{i=1}^{n} \delta(w_i) \le K$

(c5) $\sum_{i:w_i > Lb} w_i \le Ub$

(c6) $\mathbf{Aw} \le \mathbf{b}$ and $\mathbf{A}_{eq}\mathbf{w} = \mathbf{b}_{eq}$

where

$t=1,\dots,T$	time period
n	number of available assets (composing the index)
$R_t^{\,B} = \ln(\frac{B_t}{B_{t-1}})$	index (benchmark) log-return at time t
B_t	index value at time t
$R_t^{\,P} = \mathbf{r}_t\mathbf{w}$	portfolio return at time t
$\mathbf{r}_{Txn}=[\mathbf{r}_t]$	Txn matrix of asset log-returns ($t=1,\dots,T$)
$\mathbf{r}_t$	1xn vector of asset log-returns at time t
$r_{t,i} = \ln(\frac{S_{t,i}}{S_{t-1,i}})$	i-th asset return at time t
$S_{t,i}$	stock price of the i-th asset at time t (i=1,...,n)

w_i	portfolio weight $0 \le w_i \le 1$ of the *i*th asset
$\mathbf{w}_{nx1}=[w_1, ..., w_n]^T$	tracking portfolio weights vector
$\mathbf{wb}_{nx1}=[wb_1, ..., wb_n]^T$	benchmark(index) portfolio weights vector
ε_i, ξ_i	lower and upper bounds for individual weight of the *i*th asset
L, K	lower and upper bounds for the number of asset positions in the replication
Lb	lower threshold for characterizing asset weights as "very large"
Ub	maximal percentage of the sum of "very large" asset weights
$\mathbf{A}_{n.ineq.constraints \; x \; n}$	coefficient matrix for 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
$\mathbf{A_{eq}}_{\; n.eq.constraints \; x \; n}$	coefficient matrix for equality constraints
$\mathbf{b_{eq}}_{\; n.eq.constraints \; x \; 1}$	vector of equality constraints

If the benchmark composition, $\mathbf{wb}=[wb_1, ..., wb_n]^T$, is known and constant over the time period, we could re-state the index replication goal as to obtain an optimal portfolio, $\mathbf{w}=[w_1, ..., w_n]^T$, with respect to minimization of tracking error volatility of a benchmark (**wb**), i.e.,

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

where $\mathbf{\Sigma}$ is the covariance matrix of the asset returns.

This specification allows to further disentangle the econometric and optimization problem and to better test for different covariance matrix estimators. We also use objective function (2) to compare the DECS-IT algorithm to QP in some test cases.

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

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

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

Formulation (3) allows us to incorporate the fact that fund managers happily accept positive deviations from the benchmark. The problem could also be re-formulated as a multi-objective optimization problem and still be successfully tackled with DE (Krink and Paterlini 2007).

From an optimization viewpoint, the main challenge in index tracking does not usually arise from the objective function, which often is quadratic, but rather from asset *selection* (constraint c4) and other non-linear constraints, such as (c5) the UCITS concentration limits, for example specify *Lb*=5% and *Ub*=40% as binding for most portfolios in the EU (German Investment Law §60). Asset *selection* means that only a certain number of assets of the benchmark are used for replication, as formulated as constraint (c4). This means that the number of non-zero weights is bounded, which is an integer constraint and hence non-linear[2].

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 the presence of constraints (c4) and (c5), in particular the issue of (discrete) asset selection while searching for (continuous) asset allocation weights, and because of the high dimensionality.

In principle, integer constraints, like (c4), can be tackled by applying QP and branch and bound search (BB) to a reformulated version of the problem. Here, 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 via BB. For each sub-problem (c4) is replaced by constraints that define one (fixed) choice for asset selection. 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 selection. The search space for BB, however, would be extremely large for realistic problems. For instance, for a maximum of 50 asset positions out of 225 in the benchmark, one needs to search among all possible combinations of 50 positions in 225, which amounts to 6.4443e+52 QP sub-problems. Even more problematic is constraint (c5). Even theoretically it cannot be tackled with BB, because it cannot be divided into a discrete number of sub-problems.

In view of this, the combination of BB and QP procedures is not applicable. Less restrictive search heuristics, such as the one we propose here, are required for the problem at hand.

---

[2] 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 describe 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 do not require restrictive properties, such as continuity, linearity or convexity of the objective functions and the constraints. Hence, evolutionary algorithms can easily deal with cardinality constraints, non-linear constraints and other real-world restrictions when implementing a quantitative index tracking.

Differential Evolution (Storn and Price 1995, Price 1999) is a fairly novel population-based search heuristic which is simple to implement and requires little parameter tuning compared to most other search heuristics. Differential Evolution has shown superior performance in comparison to other heuristics, such as Genetic Algorithms and Particle Swarm Optimization, when 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 DE 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 this direction and show how DE combined with other operators can provide reliable and robust results in index tracking.

Differential Evolution is a population based search heuristic. After generating and evaluating an initial population, ***P***, of candidate solutions, the candidate solutions are iteratively refined as follows:

1. 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 scaling factor, f, and add the result to $\boldsymbol{P}(m)$, i.e.,

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

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

3. evaluate the new candidate solution with the fitness function and apply selection by substituting ***c*** for ***P****(j)* (and updating the fitness of ***P****(j)*) if ***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 simultaneously tackle the continuous numerical and combinatorial aspects for a very large number of parameters. 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 the n-dimensional space $[0,1]^n$ and therefore all solution components are continuous numbers. However, despite DE being a very effective algorithm for continuous numerical problems, our preliminary experimentation showed us that it may fail in index tracking when used without modifications. Therefore, we implemented an additional operator, that we call "position swapping", which swaps positions by exchanging the asset weights between two assets with a zero and a non-zero allocation with a certain probability. With introduction of this operator the quality of the results and the required runtime could be substantially improved. This does not surprise considering that selecting asset *j* or *k* implies that w_j is zero and w_k is redistributed to other weights or vice versa.

3.3 Constraint Handling

As stated 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,[3] 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 (see Michalewicz and Fogel 2000

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

for an overview). The main idea of penalty techniques is to re-state 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, $P(\mathbf{x})$ is zero.

The penalty term can be defined in various ways. It could depend on the number of constraint violations, the violation sizes or on runtime. The drawback of the penalty approach is its parameter sensitivity, such as the chosen size of a penalty (Deb et al. 2002).

In case of index tracking, some constraints, such as (c1) and (c3), can be handled 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. The remaining constraints can be handled 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 constraints less severely than the other
                   select the solution that violates less severely
            else
                   select the one with better fitness
            end
     end
end
```

3.4 Initialization

An issue that is often neglected in designing optimization heuristics is the initialization of the process. A common strategy is to use random initialization, which generates a population of candidate solutions that 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 assets.

Perhaps the most obvious consideration is to avoid positions that are highly correlated, because they can be virtually expressed by one another. Instead, one should rather choose a set of least correlated asset selections to have the maximum flexibility for replicating all aspects of the assets that are included in the benchmark. To do so, one could first cluster the assets based on their correlations and then pick representatives of each cluster. Focardi and Fabozzi (2004), for instance, suggest that positions should be strictly chosen in accordance with this approach. However, there are many factors 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. Below, we will cluster the log-returns time series with hierarchical clustering using 'Ward' linkage and setting the upper bound for the number of asset positions to be selected, K, as the number of groups to obtain the final data partition. From each group we select the asset 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 of replicating an important asset is to use the asset itself. This is easily implemented if the index composition is known at the beginning of the period. If not, market values or other values could be used as proxies, depending on whether the index is value-weighted or not. The idea there is sort the index components by their weights and to select the first K asset positions. In particular assets with very large 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, replicating the Nikkei 225 index with 30 assets, the index weights computed from market values on 12/01/2005 are, w_{30} = 0.00743, for the asset that ranks 30 in the top-n largest weight ranking, and w_{31} = 0.00737, for the asset with rank 31. Hence, the preference for the former position is almost arbitrary.

We compare the three initialization methods for our algorithms, calling them *random*, *top-K least-correlated* and *top-K largest weights*. Empirical results are reported in Section 4.3.3.

4 Experiments and Results

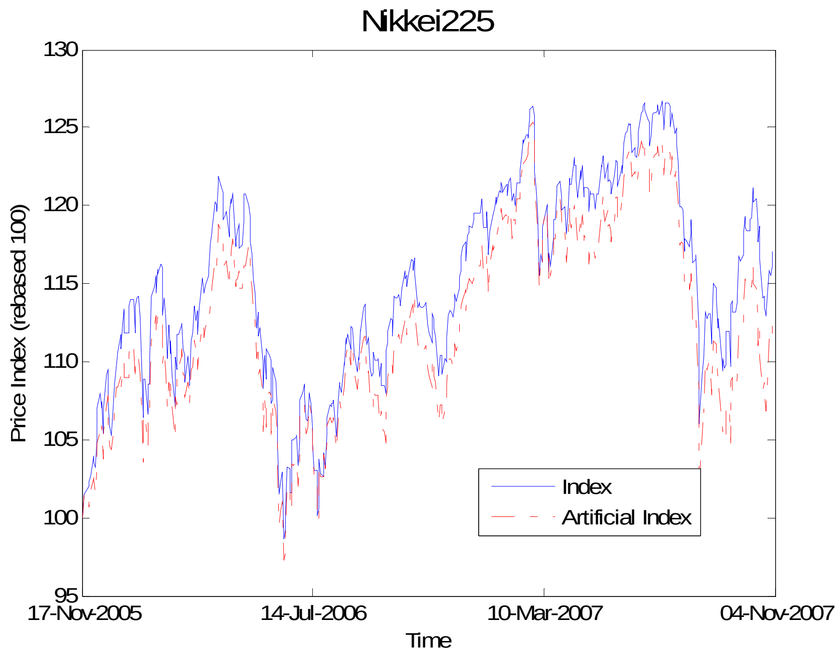
In the following, we describe our experiments, discuss runtime performance and compare the algorithm with QP on test cases. Finally, we also discuss results for different seed initialization schemes. The reader is referred to Appendix A for a description of our preliminary experiments on parameter tuning.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

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 problem specification

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 its 225 constituents for the sample period 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 (blue solid line) and artificiaL Nikkei 225l benchmark (red dashdot line), scaled to 100 on 17/11/2005



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 results with high precision 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 preliminary experiments to investigate three issues:
(i) the ability of DECS-IT to fine tune asset weights,
(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 adopting objective function (2) and excluding constraints (c4)-(c6). For experiment (i), we investigate the weight fine-tuning ability of our algorithm independently of the asset selection. For comparison with QP, we simplify the original problem to a quadratic programming problem.

The parameter settings for the optimization algorithms are as shown in Table 1.

4.2.1 Weight Fine-tuning Ability

In the first set of experiments, we test our algorithm for its general ability to numerically fine-tune the asset weights. For this task, we simplify the problem, such that there are no constraints on the number of asset selected. Clearly, the optimal solution is **w** = **wb**[4]. However, for a search heuristic starting with a population of random candidate solutions, the algorithm has to fine-tune all 225 weights precisely to match the correct result, i.e., the true index weights.[5]

Table 2 shows the results for 30 runs, starting from random seed initialization (see Section 3.4). It is evident that DECS-IT is capable of approximating the correct result with high precision.

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

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

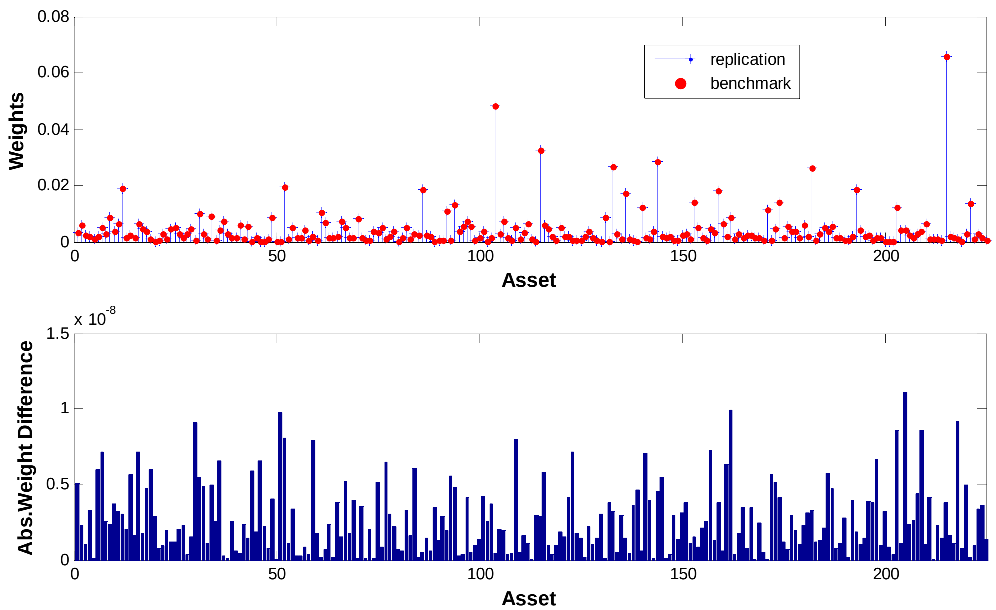
Figure 3 shows a comparison between the actual index weights and those of the tracking portfolio (top) and the absolute value of the difference in the asset weights between the index weights and the tracking portfolio (bottom). The difference in asset position weights is negligible with order of magnitude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

---

[4] 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 decimals.

[5] 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 so that the algorithm would not need to search at all.

Figure 3: Results of the weight fine-tuning experiment starting from random weights at initialization. Top: Benchmark weights (red dots) and the optimized replication (black candle sticks) for the final result. Bottom: Absolute difference between index weights and replicated weights.



4.2.2 Comparison of DECS-IT and QP

In the second test case, we compare the performance of the new algorithm to conventional optimization techniques, specifically to QP. If the index weights are known, the objective function is quadratic, as in (2), 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 under consideration. To make the comparison, we simplify the original problem to a quadratic programming problem. For this, we omit the task of finding the optimal subset selection of assets (constraint c4) and do not consider constraint (c5). We run two set of experiments:

i) We fixed a priori the selection of K=50 positions (<u>not</u> optimized) and dif not consider any lower/upper bound on the tracking portfolio weights (i.e., $\varepsilon=0$, $\xi=1$).

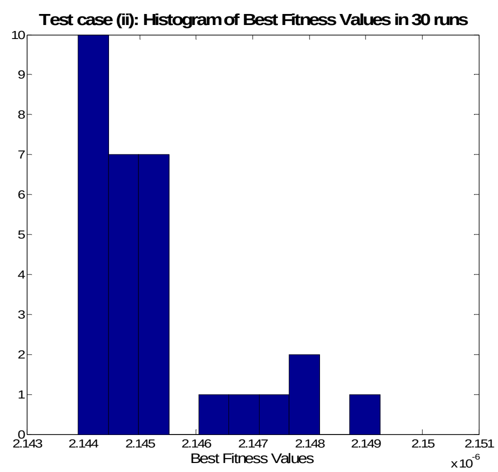
ii) We fixed a priori the selection of K=100 positions (<u>not</u> optimized) and set $\varepsilon=0$ and $\xi=\min(1,\mathbf{wb}+0.01)$.

Table 3: QP and DECS-IT comparison on test cases. QP result and Minimum, Mean, Maximum, Standard Deviation and 90% Percentile of best fitness values from 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 results comparable to 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, differences in asset weights are negligible and would not have any measurable effect in a real-world application. In fact, assets weights are not truncated to a given number of decimal places and differences have, at most, an order of magnitude of 10^{-4}, which would hardly determine any difference in the number of shares per asset bought.

Figure 4: Histogram of DECS-IT best fitness values for 30 runs for test case (ii).



4.2.3 DECS-IT Performance for Different Initializations

Next, we investigate the effect of different seed initialization. We set the constraints such that the 225 asset of the benchmark have to be replicated with a maximum of 50 positions. Specifically, we set ε=0.00, ξ=0.1, Pop.size=50, Num.It=10000, CR=0.3, f=0.7. Again, we use the NIKKEI 225 price index and its 225 constituents, considering the period 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 introduced in Section 3.4, considering:

(a) *top-K largest weights* - asset positions with the 50 largest weights,

(b) *top-K least-correlated* - asset positions of the 50 least-correlated asset clusters,

(c) *random* - one random selection of 50 assets.

Figures 5(a), 5(b) and 5(c) show the results from single runs of DECS-IT for the three initialization schemes. The left plots show the tracking portfolio weights (blue 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. Ursem (2002) defines diversity as the sum of the variance of the population divided by the size of the population and the diagonal of the search space.[6]

As expected, DECS-IT obtains the best result with $f(\mathbf{w})$ = 5.6898e-7, for the top-K largest weights initialization. The rights plots show that 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 explores 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 as 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, again, to increase sharply 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.

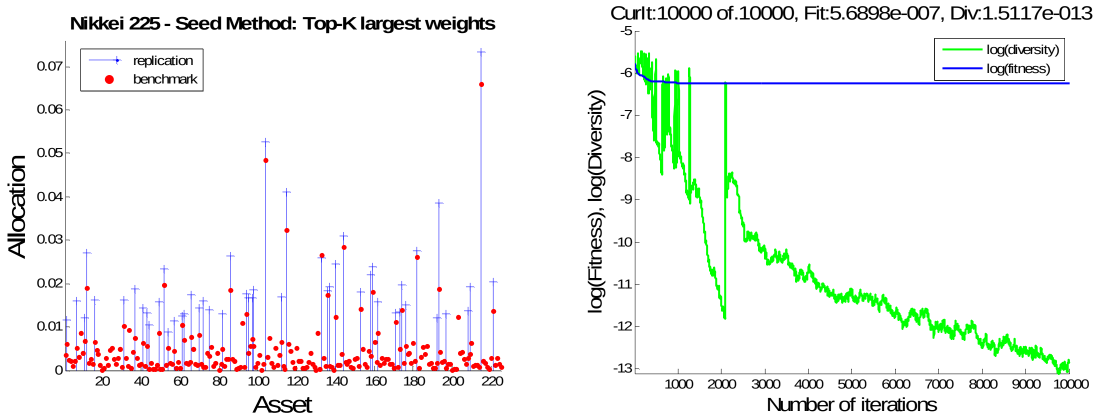
Finally, Figure 6 shows the boxplots of the best fitness values in 30 runs reached for objective function (1) (left plot) and objective function (2) (right plot) for K=50, ε=0.00, ξ=0.1, Pop.size=50, Num.It=10000, CR=0.3, f=0.7. The *top-K largest weights* initialization seed scheme is to be preferred for both objective functions. The *K-least correlated* method is generally worse than random initialization for objective function (1). To consider the K largest market values as starting 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 $\mathbf{\Sigma}$ estimates) does not seem to be advantageous for DE initialization.

---

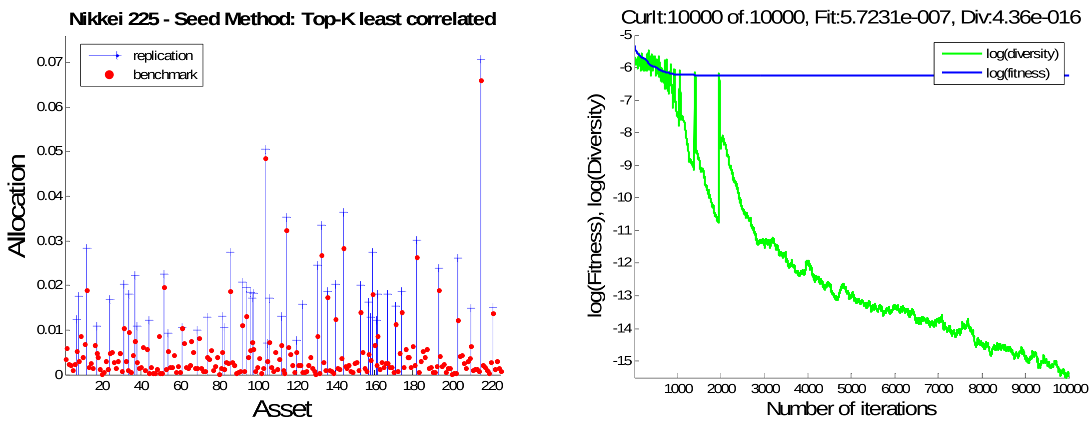
[7] I.e.: div = sum(Variance(pop)) / (Pop.Size * L); L = sqrt(sum((ξ-ε).*(ξ-ε))).

Figure 5: Results of the DECS-IT experiments for 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 for the final result. Right: log best fitness (blue) 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

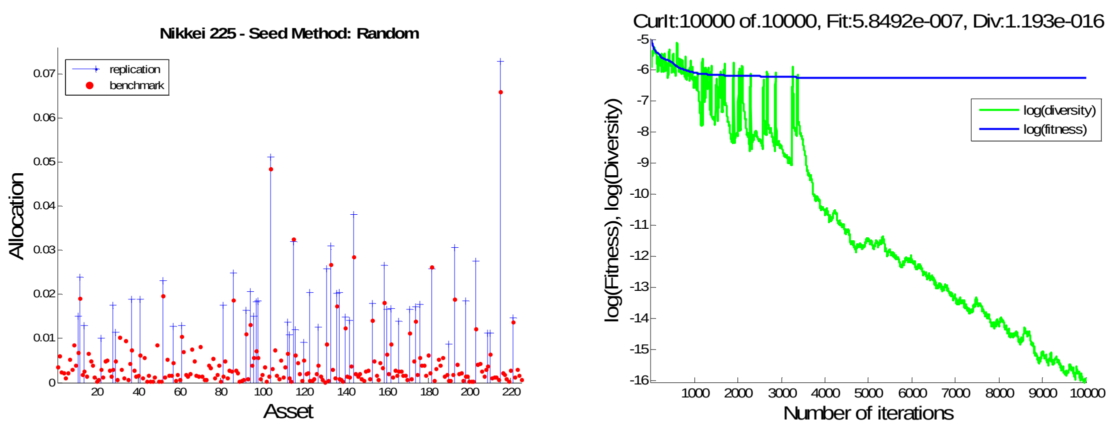
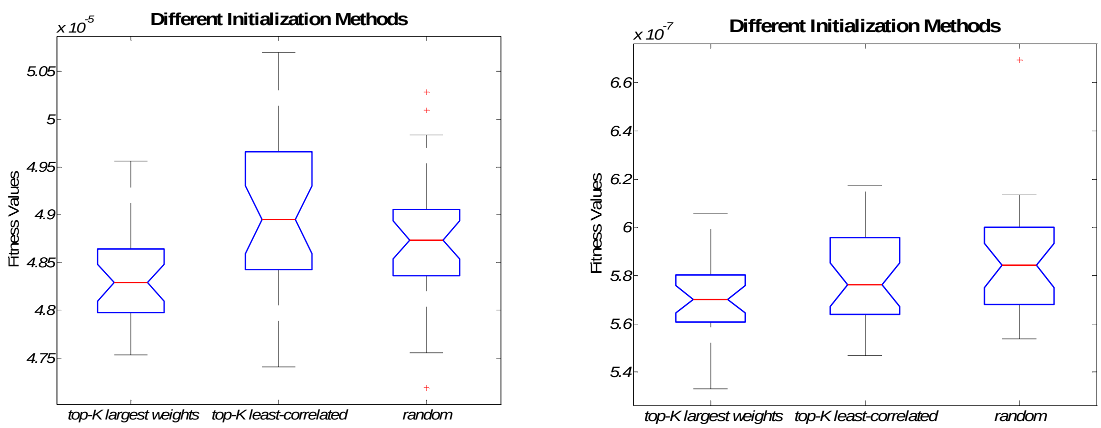


Figure 6: Boxplots of best fitness values for 30 runs for (a) *top-k largest weights*, (b) *top-K least-correlated* and (c) *random* initializations schemes. Left: Boxplots of the best fitness values for objective function (1), Right: Boxplots of the best fitness values for objective function (2)



5 Empirical Analysis

We now discuss empirical results from optimization over different time horizon. We investigate the in-sample and out-of-sample performance of optimal tracking portfolios for the Dow Jones 65 (Period: 13/04/02- 20/12/03) and the Nikkei 225 (Period: 18/11/05-27/07/07) indexes. Table 4 provides descriptive statistics and Figure 7 shows the behavior of the two indexes. We determine the optimal index tracking portfolios by periodic rebalancing of the assets weights. We use a rolling-window scheme: For each index, we determine the optimal tracking portfolio considering daily log-returns for window [*T, T+200-1*] and evaluate its out-of-sample performance over window [*T+200, T+200+20*] for *T* from 1 to 221 with step size 20. Hence, we altogether rebalance the portfolios twelve times.

We optimize with respect to objective function (1), considering constraints (c1)-(c5) and set the optimization parameters to ε_i=0.01, ξ_i=0.1 (i=1,…,n), *Lb*=0.05, *Ub*=0.4 and L=0. We investigate the effect of varying the maximum number of assets, *K,* that can be included in the index tracking portfolios.

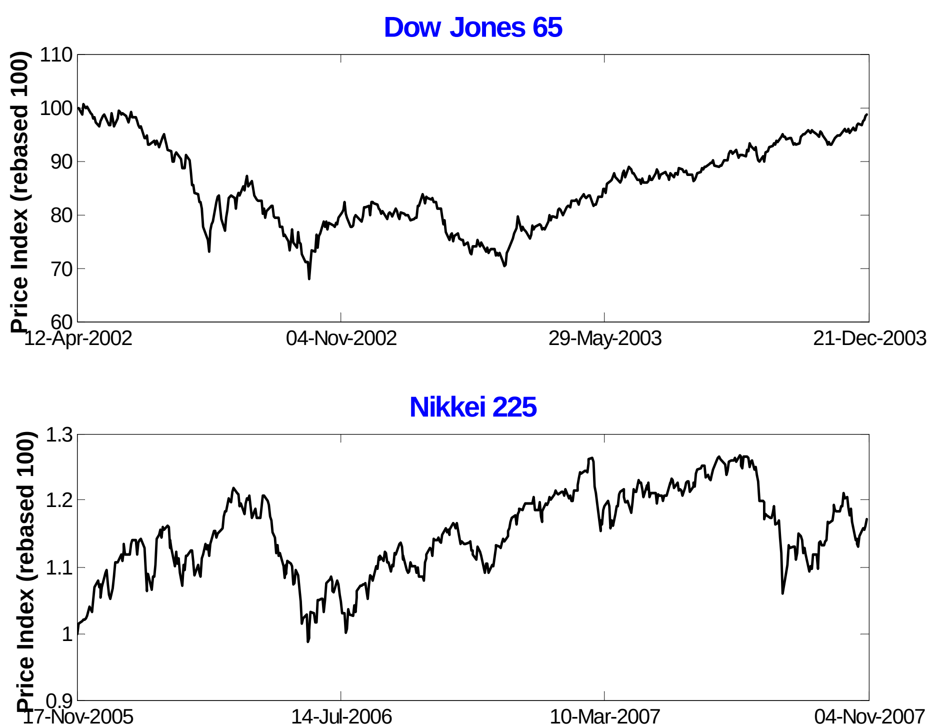
So far we did not consider transaction cost. We now introduce the constraint $\sum_{i=1}^{n} |\Delta w_i| \le 0.1$. That is, the total change in asset weights from the allocation at the previous time-step must not exceed 0.1. This constraint allows controlling for transaction costs when updating the tracking portfolio from one period to the next. The DECS-IT algorithm can easily deal with such a constraint, even if is non-linear. Without such a constraint, the changes in asset weights over two consecutive periods can be extremely large,[7] making index tracking too expensive. Transaction costs could be explicitly incorporated in the optimization task. This will be investigated in the future.

We set DECS-IT parameters to: Pop.size=100, Num.It.=10.000, CR=0.7 and f=0.3. We employ 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

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

[7] Empirical results are available upon request.

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 compute the following statistics:

- In-Sample[8] 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, computed 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)

As expected, increasing the maximum number of assets that can be included in the tracking portfolio tends to lower the in-sample annualized tracking error volatility. However, for K=55 for the 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 due to the presence of constraints. In fact, the in-sample tracking error volatility is smaller for the first 200 observations window, but the constraints on the turnover and the asset weights minimum and maximum bound lead to suboptimal portfolios in the other windows, compared K=50. In case of the Nikkei 225, we notice that even when fixing 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 also decreases monotonically (see Column 5). However, further investigation on that need to be conducted. A strategy for choosing K could be to stop the algorithm when the average in-sample tracking error volatility stops decreasing. However, this may lead to a high K which is often suboptimal when transaction and monitoring costs and other statistics are considered. In fact, we notice that there is no monotonically increasing relationship between the maximum number of assets and the in-sample and out-of-sample annualized excess returns: The maximum in-sample and out-of-sample excess returns are for K=30 and K=25 for Dow Jones 65 and for K=20 for Nikkei 225, respectively. The out-of-sample information ratio, the ratio between the out-of-sample excess return and the tracking error volatility, measures the excess return of a portfolio divided by the portfolio's risk relative to an index. The ratio jointly captures risk and return of the tracking portfolio. Columns 8 indicates that the information ration achieves maximum

[8] The in-sample annualized tracking error volatility corresponds to the average of the annualized in-sample tracking error volatilities for 12 time periods.

value for K=25 for Dow Jones and K=20 for Nikkei 225, suggesting that a portfolio with a limited number of assets should be preferred. Column 9 shows that the constraint on the total turnover is always satisfied and that the transaction costs are proportional to the 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. A correlation or beta equal one is highly appealing. Increasing K does not necessarily lead to out-of-sample correlations or betas closer to one. We have the best performance with respect to correlation and beta for K=45 for Dow Jones 65 and K=70 and K=30 for Nikkei 225, respectively.

Summing up, choosing the maximum number of assets to be included in the tracking portfolio is a difficult task. A large number of assets, apart from being potentially more expensive in term of transaction and monitoring, could lead to tracking portfolios with lower tracking error volatility (but smaller excess return), information ratios, correlations, and betas. On the other hand, a more parsimonious choice of K, could restrict diversification and result in tracking portfolios with risk-profile different from the index. This issue is currently under investigation.

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 study, we focus on the optimization problem, proposing a new heuristic that succeeds tackle in reasonable runtime. The optimization requires selecting an optimal subset of asset positions from the benchmark and finding the optimal asset weights while satisfying additional linear and non-linear constraints. A further complication arises we deal with high-dimensional problems. Although the objective function for the index tracking problem could often be stated as a quadratic function (2) it is not a quadratic programming problem, due to the presence of non-linear constraints. Therefore, quadratic programming cannot be employed. 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 failed solving this problem sufficiently well. However, we could show that an extension of Differential Evolution works remarkably well.

Our final experiments show that a good starting point for the search is simply to select the asset positions that have the largest weights in the benchmark, though random initialization can also generate meaningful results. This makes particular sense for the largest weights, but less so for smaller weights, especially around the cut-off between asset positions, that should and should not be included. The latter typically have very similar weights and thus, may lead to arbitrary selections. This simple approach turned out to be slightly better than using asset positions that are least correlated and the random selection of positions. Regarding the use of least correlated positions, it should be pointed out that we only investigated one out of many possible settings. A combination of selecting the largest and the least correlated positions may also work if assets with similar weights are selected according to their correlation. We did not investigate this further, since the results from largest weight 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, QP-type formulations 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 this study, we tested DECS-IT with two different objective functions. DECS-IT could easily deal with more complex objective functions and other non-linear constraints (for example on total turnover, as discusses in Section 5).

Finally, we tested DECS-IT considering the index tracking problem over different time periods for two equity indexes, the Dow-Jones 65 and Nikkei 225. We found that increasing the maximum number of assets to be included in the tracking portfolios decreases the in- 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). This strongly depends on the index composition.[9] 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 returns, information ratios, correlations or betas.

Further research on different indexes and with different portfolio optimization settings are under investigation. As is the question of how different covariance estimators can affect index tracking.

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

Acknowledgements
Sandra Paterlini conducted part of this research while visiting the Center for Quantitative Risk Analysis and the School of Mathematics, University of Minnesota. Financial support from the MIUR PRIN 20077P5AWA005 and from Fondazione Cassa di Risparmio di Modena for ASBE Project is gratefully acknowledged.

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.

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., Filipiˇc, 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) subject to constraints (c1) – (c4) with ε=0.01, ξ=0.10[10], 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 DE 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 the DECS-IT algorithm. Figure A.1- A.4 show the main results of our investigation.

In realistic applications, such as index tracking, the computational bottle-neck of search heuristics is the evaluation of the objective function. 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 sizes yield better results if the number of evaluations is large enough (e.g., 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-value distributions for a scaling factor equal to 0.3 and 0.4, while a scaling factor equal to 0.2 leads to slower convergence. Figure A.2 (b) reveals that the choice of the scaling factor tends to affect convergence less as the number of iterations increases.

---

[10] Note 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.

This 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 is usually required for converging 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 convergence, as the three boxplots have similar distributions for the best fitness values. Furthermore, Figure A.3 (b) points out, as expected, that the number of iterations is the crucial parameter. In fact, increasing the number of evaluations from 2500 to 10000 leads to a more steadily converge towards lower (best) fitness values and less dispersion, 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 convergence independently of the crossover rate chosen. Our results suggest that a scaling factor equal to 0.2 could be too low, even for 10000 iterations, while all other parameters do hardly alter the quality of the results. However, it should be noticed the order of magnitude of the difference in the best fitness is very small. The reader is refereed to Krink and Paterlini (2007) and to Krink et al. (2007) for further evidence on 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};

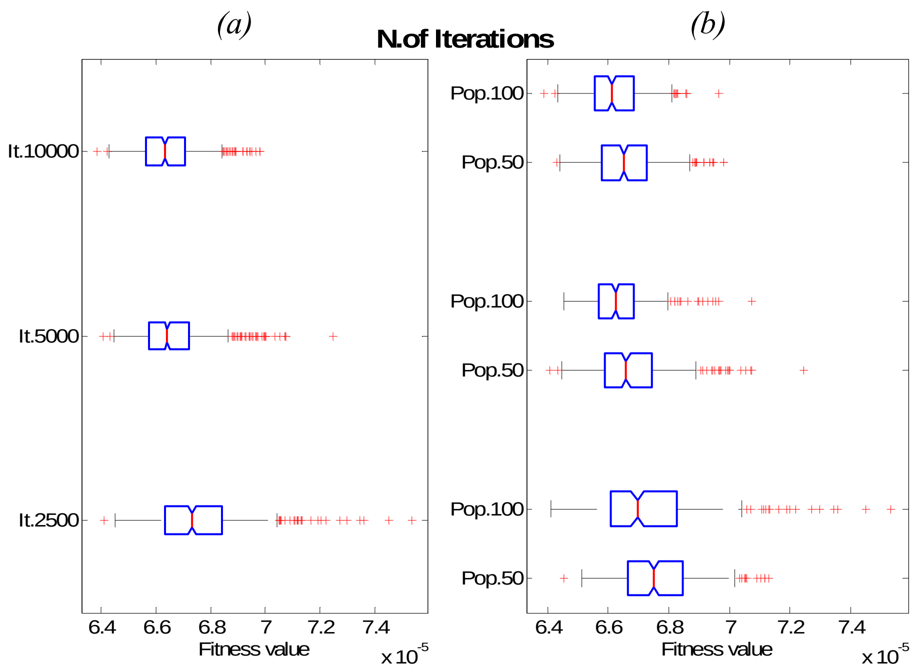


Figure A.2: (a) Boxplots of best fitness values for scaling factor {0.2, 0.3, 0.4} for 30 runs for each combination of the following parameter settings: number of iterations {2500, 5000, 10000},population size {50, 100}, crossover rate {0.7, 0.8, 0.9}; (b) Boxplots of best fitness values for scaling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

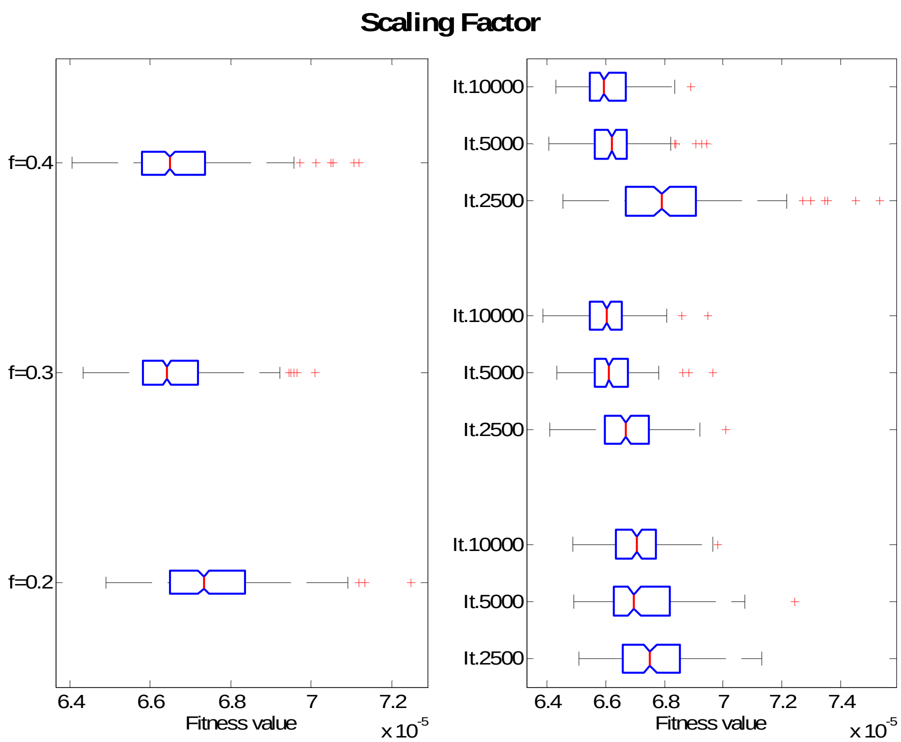


Figure A3: (a) Boxplots of best fitness values for crossover rate {0.7, 0.8, 0.9} for 30 runs for each combination of the following parameter settings: number of iterations {2500, 5000, 10000},population size {50, 100}, scaling factor {0.2, 0.3, 0.4}; (b) Boxplots of best fitness values for crossover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

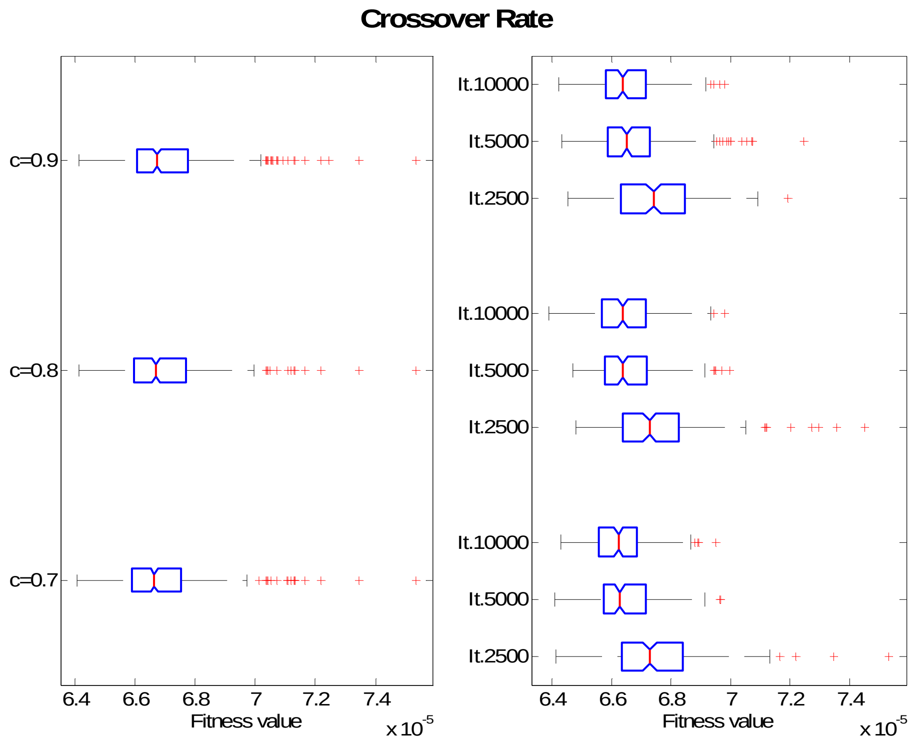
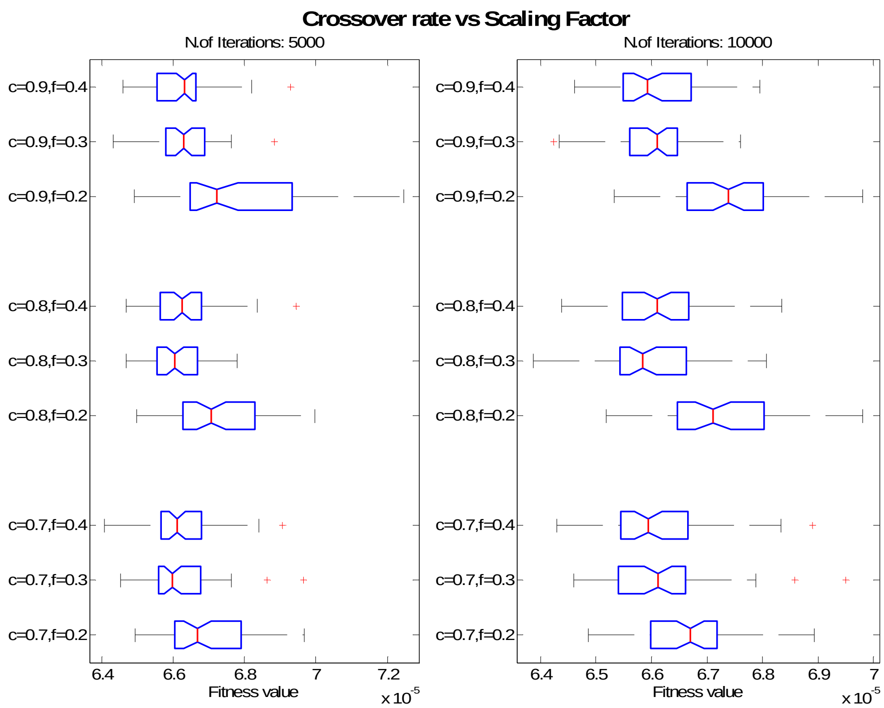


Figure A.4: (a) Boxplots of best fitness values in 5000 iterations for crossover rate {0.7, 0.8, 0.9} and scaling factor {0.2, 0.3, 0.4} for 30 runs for each combination of the following parameter settings: population size {50, 100}, crossover rate {0.7, 0.8, 0.9}, scaling factor {0.2, 0.3, 0.4}; (b)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

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

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
```