ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Semester Project
Department of mathematics

Mixed integer programming applied to tracking error optimization in active portfolio management

Laurent Gorgé

Supervision team of the
Swiss Federal Institut of Technology in Lausanne:
Prof. Thomas M. Liebling
and
Frank Crittin

Supervision team of UBS - BRINSON in Zurich:
Dr. C. Kessler
Dr. G. Schwarz
Dr. F. Härtel

Spring 1999

Contents

Introduction.

Tracking error optimization in portfolio management is an important field of research. Since the creation of indexed funds, the need of an algorithm for maximizing the correlation between the positions of the fund and the benchmark is born. But this problem is not trivial for really big benchmarks, such as the index S&P 500, which takes 500 assets in consideration.

An efficient algorithm for smaller markets was done by M. Kämpf in his diploma thesis at the Swiss Federal Institute of Technology (SFIT) in February 1997. Our goal in this study, was to try to use this diploma thesis and find a way to accelerate it to get solution for bigger markets in a reasonable time. We try to get an algorithm which gives us a good approximation.

This problem was given to the SFIT in Lausanne, by UBS Brinson. I would like to thank the supervision team at the SFIT, for there interesting comments. I also would like to thank the Supervision team at UBS Brinson for the constructive meeting we had.

In the first chapter we would recall some of the principal elements of financial mathematics, then in chapter two, we will formulate mathematically our problem. The chapter three is explaining the "Branch & Bound" algorithm. This algorithm is modified in chapter four, and the results of this modification state in chapter five. After the conclusion the listing of the implementation on Matlab is given in appendix.

Chapter 1

Elements of financial mathematics

1.1 The Mean-Variance criterion

The theory of portfolio management is based on a simple normative criterion, developped in the 50's by Professor Harry Markovitz. This criteria is known as the mean-variance criterion.

Definition 1 :
The **Mean-Variance criterion** *stipulate that the investor must, or at least should, think in terms of the mean and the variance of the expected return of his portfolio. He should choose, for a given level of expected return, the minimum variance portfolio. Or inversely, he should, for a given level of variance, choose the portfolio which maximize the expected return.*

The return of a portfolio beeing defined as:

Definition 2 :
The **expected return of portfolio** P *equals the weighted sum of the expected returns for each assets :*

$$E[R_P] = \sum_{i=1}^{N} X_i E[R_i]$$

where X_i *is the weight of the* i^{th} *asset in the portfolio,* R_i *the return of asset* i *and* R_P *the return of portfolio* P*.*

The portfolio's variance represents a mesure of the return dispersal around his mean and thus can be assimilate to an objective mesure of the risk.

Unlike the return, the variance of a portfolio does not simply equals the weighted sum of the assets' variances. It depends also on the interdependance between the assets' returns. That's why the covariances between the assets should be used.

Definition 3 :
The **portfolio variance** *is defined as:*

$$\sigma_P^2 = \sum_i \sum_j X_i X_j \sigma_{ij}$$

Where σ_{ij} is the covariance between the return of asset "i" and the return of asset "j".

1.2 Diversification

Definition 4 :
We call **diversification** *the fact of increasing the number of assets in a portfolio.*

The effect of diversification is a decreasing portfolio variance, because of the increasing space in which we pick the portfolio. That means for each assets we add in the portfolio, the set of all portfolio we can make with these assets is growing, and thus, there can be a portfolio with a smaller variance than before.

1.3 Efficient frontier

It's theoretically possible to represent all available assets and all possible combinations of these assets in risk-return space. The result would be a cloud of point as in Figure 1.1. The efficient frontier is the upper convex hull of this cloud of points.

We assume that every investor prefers more return than less for a given risk, as well as less risk to more for the same return. By example, on Figure 1.1, an investor would prefer portfolio B to A if he is interested in having a risk equal to V2. If he is interested in portfolios with a return equal to R1, he would prefer the portfolio C to A.

1.4 Portfolio optimization

The goal in portfolio optimization is to track a benchmark or a strategy with a limited number of assets. The smaller the number of assets the better it is, since each transaction has a cost.

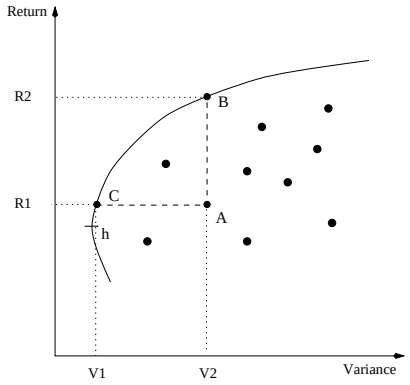

Figure 1.1: Efficient frontier.

Given a fixed number of assets, portfolio optimization is then finding the best portfolio in term of Mean-variance criterion in the set of all portfolio strongly correlated to the benchmark. It's in fact a compromize between the number of assets and the correlation with the benchmark. The more assets we buy the bigger the cost are, but also the bigger the correlation is. Obviously, if you trade all the assets you'll have a correlation equal to 1.

Chapter 2

Problem formulation

2.1 General formulation

A typical problem in portfolio management is the one which transcript the mean-variance criterion. It is to minimize the variance of the portfolio, under the constraints of full investment and given a fixed expected return. Mathematically we can represent a portfolio by a N-vector $\vec{w}$ in which the components are the weight of the corresponding assets in the portfolio, with this, the problem becomes:

$$\min \vec{w}^T V \vec{w} \tag{2.1}$$

$$\text{s.c. } \vec{w}^T \mathbf{1}_N = 1 \tag{2.2}$$

$$\vec{w}^T f = f_P \tag{2.3}$$

Where

$$\begin{aligned} V &= \text{Variance-Covariance Matrix,} \\ \vec{w} &= \text{Portfolio's weights,} \\ f &= \text{assets expected return,} \\ f_P &= \text{Optimal portfolios expected return ,} \\ \mathbf{1}_N &= \text{Unitary vector of length } N, \\ N &= \text{Total number of assets in the market.} \end{aligned}$$

Our problem of interest is quite similar. Here, the aim is to minimize the tracking error between an active portfolio and another portfolio such as an optimal strategy, with the request that both portfolio should have the same expected return. We also have the constraints of full investment and we choose to change only a given number of assets. Moreover we are not allowed to short positions. We will take a benchmark as active portfolio, but this is not restrictive.

The data we know are:

- Benchmark portfolio B with weights $\vec{w}_B$,
- Optimal strategy portfolio S with weights $\vec{w}_S$,
- Information Ratio of strategy S: IRs (ratio of anualized expected residual return to residual risk),
- variance-covariance Matrix V (positive definite),
- Number n of assets we allow us to change,
- The expected return of the resulting portfolio f_P.

The problem becomes:

$$\begin{aligned} \min \quad & (\vec{w}_B - \vec{w}_S)^T V (\vec{w}_B - \vec{w}_S) && (2.4) \\ \text{s.c.} \quad & \vec{w}_B^T \mathbf{1}_N = 1 && (2.5) \\ & \vec{w}_B^T f = \vec{w}_S^T f && (2.6) \\ & \#\text{ changes} = n && (2.7) \end{aligned}$$

The constraint 2.5 shows the full investment. The constraint 2.6 shows the fact that the resulting portfolio should have the expected return f_P.
The constraint 2.6 is often replaced by the fact that the resulting portfolio and the Strategy should have the same β.
The constraint 2.7 represent the fixed number of changes allowed.
β beeing defined as:

Definition 1 :
β is the constant which mesure the sensibility of the portfolio to the market.

By example, if $\beta = 1$ we have a portfolio which gives exactly the same return as the market. We are therefore going to use the constraint that both portfolio should have the same β instead of constraint 2.6.

2.2 Tracking error optimization

This method evaluates the difference between the portfolios by mean of the squared error.

Definition 2 :
The **Tracking error** *between the portfolio P and the portfolio S is defined by:*

$$\begin{aligned} TE(P,S) &= \sqrt{(\vec{w}_P - \vec{w}_S)^T V (\vec{w}_P - \vec{w}_S)} \\ &= \sqrt{\vec{w}_P^T V \vec{w}_P - 2\vec{w}_P^T V \vec{w}_S + \vec{w}_S^T V \vec{w}_S} \end{aligned} \tag{2.8}$$

We remark that the problem doesn't change if we minimize the squared Traking error since the square root is a strictly growing function.

Definition 3 :
Let $TEP(n)$ be the **resulted portfolio** *with n changed positions in the active portfolio and with full investment.*

More mathematically, the problem becomes :

$$\begin{array}{ll} \min & (\vec{w}_{TEP(n)} - \vec{w}_S)^T V (\vec{w}_{TEP(n)} - \vec{w}_S) \\ \text{s.c.} & \vec{w}_{TEP(n)}^T \vec{\beta} = \vec{w}_S^T \vec{\beta} \\ & \vec{w}_{TEP(n)}^T \mathbf{1}_N = 1 \\ & \exists \vec{x} \in \{0,1\}^N \text{ with} \\ & \qquad \vec{x}^T \mathbf{1}_N = n \\ & \qquad \vec{W}_{TEP(n),i} = \vec{w}_{B,i} \quad \forall i \text{ with } x_i = 0 \end{array} \tag{2.9}$$

Chapter 3

The Branch & Bound algorithm

The so called "Branch & Bound" method is a combinatoric optimization method dealing with integer numbers. It is a tree structured method which uses the separation and evaluation of the problem in more constraint ones. This method is exploring all the admissible solutions (at least implicitly) and thus give the exact optimal solution. Let us study more deeply this method:

Definition 1 :
Let (T), be the following problem:

$$\begin{aligned} Min \quad & \vec{C}\vec{x} && (3.1)\\ s.c. \quad & A\vec{x} \leq \vec{b} && (3.2)\\ & \vec{x} \geq 0, \vec{x} \in \mathbb{N} && (3.3) \end{aligned}$$

where A is a matrix and $\vec{C}$, $\vec{b}$ some vectors related to the problem. (T) is a typical linear optimization problem. We will see later that our problem would be on this type.

3.1 The Branching

We are going to deal with the separation of our problem in subproblems. In fact, we are going to construct a tree, in separating our problem at each node in subproblem, in taking only a subset of the space of solution. The more deeper we would go in the tree, the smaller the set would be.

Definition 2 :
Let X be the set:

$$X = \{x \in \mathbb{R}^n \mid A\vec{x} \leq \vec{b}, x \geq 0, x \in \mathbb{N}\}$$

It is direct to see that the condition $\vec{x} \in X$ is equivalent to 3.2 and 3.3.
With this definition, (T) becomes:

$$\text{Min} \quad C\vec{x} \tag{3.4}$$
$$\text{s.c.} \quad \vec{x} \in X \tag{3.5}$$

We then build a tree. At each node, we let X' be a set of constraint such as $X' \subset X$.
For the node with the subset X', we will have the following problem :

$$\text{Min} \quad \vec{C}\vec{x}$$
$$\text{s.c.} \quad \vec{x} \in X'$$

Definition 3 :
Each of the node has a number p of sons. Let us now define X'_i $i = 1...p$**, a recovery of** X' *, such that* $\cup_{i=1}^{p} X'_i = X'$ *and* $\cap_{i=1}^{p} X'_i = \phi$.

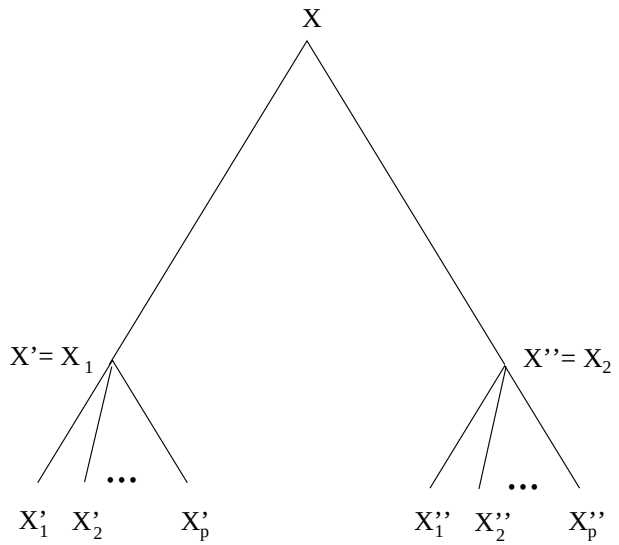

Figure 3.1: Branching on X.

We then consider p subproblems, each of them with the constaints X'_i (See Figure 3.1). The decomposition of X' in $X'_1, X'_2, ..., X'_p$ is called the branching.

3.2 The Evaluation

The problem at this point, with our tree, is that it is not realistic at all to enumerate all the admissibles solutions. Thus we should select subtrees which are not interesting to visit. It means that we have to find subtrees in which we

are sure not to find the optimum. That's why, for each node p, we calculate a lower bound $f(X')$ for the optimal value in the subtree which has p as root (and X' as constraints set). That means we search $f(X')$ such that:

$$f(X') \leq \begin{array}{ll} \text{Min} & C\vec{x} \\ \text{s.c.} & \vec{x} \in X' \end{array}$$

Definition 4 :
$f(X')$ *is called the* **bound**.

Let now $\bar{f}$ be the best known solution. Every time that we reach a leaf (admissible solution), we evaluate $f(X')$. If $f(X') < \bar{f}$ then we set $\bar{f} = f(X')$ and keep this solution as the best known. Each time we come to a node where $f(X') \geq \bar{f}$ we renounce getting further in the corresponding subtree, because :

$$\left\{ \begin{array}{ll} \text{Min} & \vec{C}\vec{x} \\ \text{s.c.} & \vec{x} \in X \end{array} \right. \leq \bar{f} \leq f(X') \leq \left\{ \begin{array}{ll} \text{Min} & \vec{C}\vec{x} \\ \text{s.c.} & \vec{x} \in X' \end{array} \right.$$

And so we already have a solution better than all the ones we could find exploring the corresponding subproblemes generated by the separation of X'.

3.3 "Branch & Bound" applied to portfolio optimization

To apply the "Branch & Bound" algorithm to our problem defined in 2.9, we set:

1. Objectiv function : $\vec{C}\vec{x} = \vec{\omega}^T(\vec{x})V(\vec{x})\vec{\omega}(\vec{x})$,
2. Admissibles solutions set : $X = \{\vec{x} \in \{0,1\}^N | \sum_{i=1}^{N} x_i \leq n\}$,
3. The branching is done in fixing some of the x_i to 0 or 1. In the following, we will indicate by $\vec{x}$ the fixed part.
4. Evaluation function : $TE(\vec{x}, S)$. Like in 2.8.

Where $\vec{\omega}(\vec{x})$ is the vector $\vec{\omega}$ in which we have selected only the components corresponding to the x_i which are non-zeros. The same notation is used for $V(\vec{x})$ which give us a matrix.

The algorithm is running in this way:

1. Let $\vec{x} = [\ ], \bar{f} = \infty$
2. if
   - $\sum_{i=1}^{N} x_i = n$ or $\sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{x_i=0}\}} = \mathbf{N - n}$

   $$\text{Where } \mathbf{1}_{\{\mathbf{x_i=0}\}} = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{otherwise} \end{cases}$$

   Then go to 3.

   Calculate $y = TE(\vec{x}_c, S)$. Where $\vec{x}_c = \vec{x}$ completed by ones to the size N.

   if
   - $y \leq \bar{f}$

   then,
   (a) Let $\vec{x} = [\vec{x}, 0]$ and go to 2.
   (b) Let $\vec{x} = [\vec{x}, 1]$ and go to 2.

   else : give up the search in this subtree.
3. Calculate $y = TE(\vec{x}_{adm}, S)$.

   Where $\vec{x}_{adm}$ is $\vec{x}$ completed with zeros (if $\sum_{i=1}^{N} x_i = n$)
   or ones (if $\sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{x_i=0}\}} = \mathbf{N - n}$).

   if
   - $y < \bar{f}$

   then set $\bar{f} = y$, keep the solution.

Chapter 4

Modification of the algorithm.

All the preceeding is already known and the application of the Branch & Bound algorithm was done by Kämpf in his diploma thesis. Our goal in this study was to try to accelerate the algorithm which, with Kämpf implementation takes quite a long time (pp 29 of Kämpf). By example, for 50 assets and 25 changes, it takes 20 hours to get the optimum.

We try a lot of unsuccesful ideas such as finding correlation between the matrix and some other parameters (eigenvalue, Rayleigh coefficient, etc...) and finally came to the following solution. This solution is not faster for hitting the optimum, but as we are going to see in chapter 5, it gives quite quickly an acceptable solution. However, it calculates also the optimum if we are very patient.

4.1 Idea

The main idea we get, was to visit a subtree only if it is possible to have a solution much better than the one we already have. We thus try to go in the solution set with big steps first and then as the space where to search decreases, go with smaller and smaller steps.

What we have done is using the Branch & Bound algorithm several times, decreasing the level of our requirements. That means, for the first time, we visit a subtrees only if the lower bound for this subtree is smaller than ten percent of the actual best solution (or the upper bound given in input). That gives us a solution S. At this step, we are sure that the optimal solution is not smaller than 10 percent of our actual best solution (S). Then we run the algorithm again replacing 10% by 20% etc. In fact what we do is decreasing the space of solution. (Cf 4.1 where f_i is the value of the optimal solution of step i). At any time, we have a quality control on our solution. This mean that after the j^{th} call to the Branch & Bound algorithm, the optimal solution is in $[j/10 * S, S]$.

Figure 4.1: Step decreasing solution space.

4.2 Algorithm

The algorithm is :

Input :

- (V : Variance-Covariance matrix),
- $\vec{w}_B$: Active portfolio,
- $\vec{w}_S$: Optimal strategy portfolio to track,
- f_s : Upper bound,
- IRs : Information ratio of strategy S,
- n : number of change allowed.

Output :

- $\vec{x}_{min}$: vector of changed assets,
- $\vec{w}_{min}$: vector of optimal weights after the changes,
- y_{min} : tracking error between portfolio S and $\vec{w}_{min}$.

Algorithm :

1. $s = f_s$
2. For i:= 1 to 10 do
   - percent $= i/10$.
   - applied the "percented B&B" algorithm with input:
     - s: upper bound,
     - percent,
     - V,$\vec{w}_B$,$\vec{w}_S$,IRs,n.
   - $s = s^*$. ($\vec{x}^*, \vec{\omega}^*, s^*$ = solution with the percented B&B)
   - display(s^*).
3. $y_{min} = s^*, \vec{x}_{min} = \vec{x}^*, \vec{\omega} = \vec{\omega}^*$

Where the "percented B & B" algorithm stands for the Branch & Bound with the condition "$y \leq \bar{f}$" replaced by "$y \leq \bar{f} *$ percent" in the Branch & Bound algorithm.

4.3 Comments

This Algorithm is not more difficult to transcript in any language than the original Branch & Bound algorithm. One can argue against it that it gives quickly only approximation. But what is interesting is that it will always give the optimal solution (under the assumption that we are really patient, of course). So it is not a random method which picks up some solution and keep the best one. It is an algorithm which, at each step gets a better solution.

We also remark that the algorithm can give during two or three steps the same solution. But that does not mean it was not interesting to do this step, because at each step we have a smaller bound for the error. This bound is quite big at the begining, and decreases at each step.

As our lower bound in the algorithm is really not perfect (in the sens that it can be very under estimated), we can expect to have practical bound much better than the theoretical one. So we see that a further development of this study could be to find a better lower bound for the subtrees. Another further development could be to study if another function for the choice of the "percent" would give better results.

Chapter 5

Results & Comments

On this chapter, we will show and discuss some of the relevant results we get with the preceeding method. We made the experimentation in three parts :

1. For four markets, with 20, 30, 40 respectively 50 assets we run the algorithm for different numbers of change allowed. (Cf section 5.1).
2. For a fixed number of changes allowed and a fixed number of assets, we generate 20 markets and run the algorithm. This is done in section 5.2.
3. We then try the algorithm for bigger markets. (See section 5.3)

This simulations where done on Silicon Graphics computer with a 150 MHz processor on Unix, using Version 5.2.2.1784 of Matlab.

5.1 Evolution with a growing number of changes

We try our algorithm on 20, 30, 40 and 50 assets market. For each of them we run the algorithm for a wide range of n.
The inputs where :

- The Market : the 20, 30, 40 respectively 50 first assets on the S&P 500, sorted in alphabetical order.
- The active portfolio beeing the renormalized S&P 500.
- The optimal strategy defined as $1/N$ ($N = 20, 30, 40, 50$) for each asset.
- Upper bound : $f_s = 1$.

5.1.1 20 assets market

In Figures 5.1, 5.2 and 5.3 we show the error between the optimal solution, found at the end of the algorithm, and the solutions after the iterations of the "percented Branch & Bound".

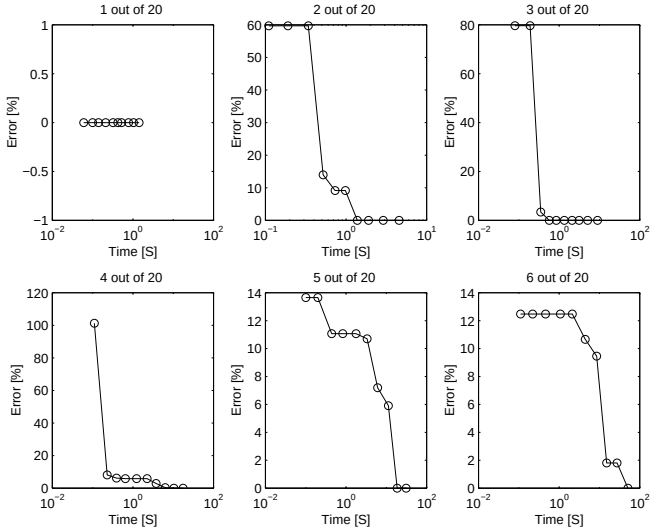

Figure 5.1: Results with the 20 assets market, for 1 to 6 changes.

On Figure 5.1, we can see that after the seventh iteration (i.e. when we have percent $= 0.7$) we have a solution which is less than 10%.

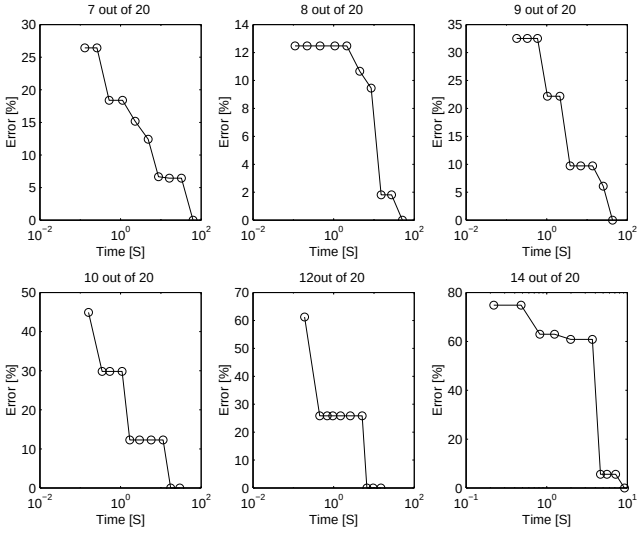

Figure 5.2: Results with the 20 assets market, for 7 to 10, 12 and 14 changes.

In 5.2, the convergence is not so fast, but that does not surprize us, since the number of admissible solutions is bigger.

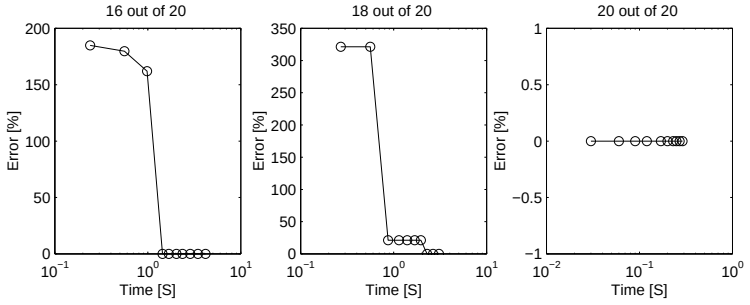

Figure 5.3: Results with the 20 assets market, for 16, 18 and 20 changes.

The results which are important are the ones with a small number of change, due to transaction costs. That is because allowing to do one more change, after a while can be more expensive than it decreases the tracking error. section In practice, such algorithm are used to update the active portfolio, an thus, it is not realistic to change a big percentage of it.

5.1.2 30 assets market

We now look at the results for the 30 assets market.

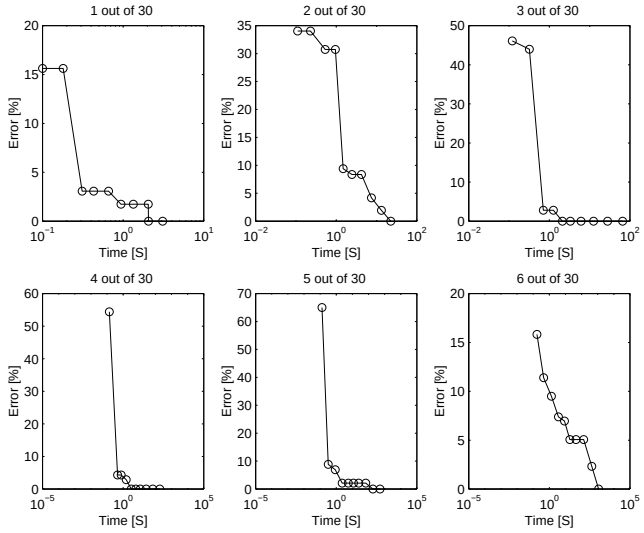

Figure 5.4: Results with the 30 assets market, for 1 to 6 changes.

Again, we see in 5.4 that the convergence is pretty fast, even faster than the one in the 20 assets market. Especially for small n (up to 9), we see that after the third iteration we already have an error less than 10 %. In 5.5 we remark that

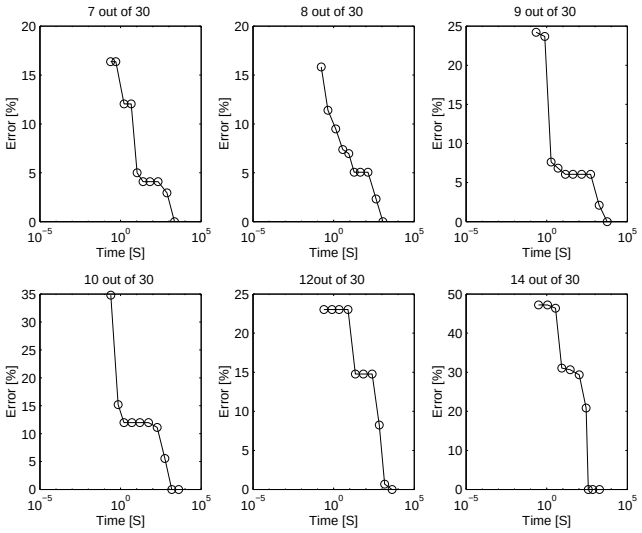

Figure 5.5: Results with the 30 assets market, for 7 to 10, 12 and 14 changes.

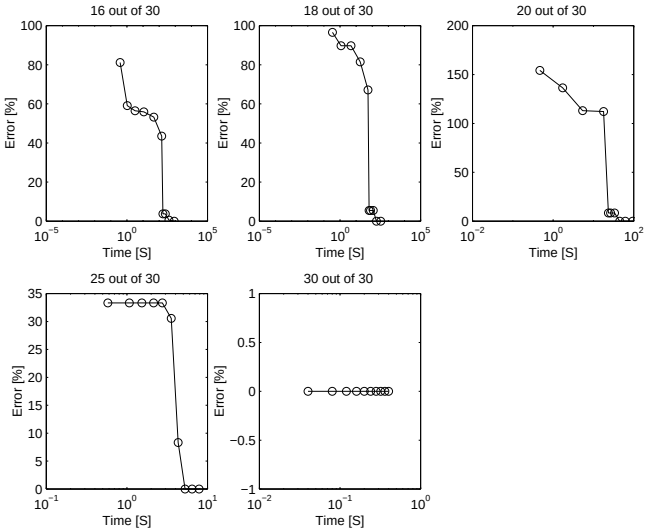

Figure 5.6: Results with the 30 assets market, for 16, 18, 20, 25 and 30 changes.

the convergence is as before slower when we take n equals to about a half of the market size. This phenomen is not surprizing, since the number of admissibles solutions equals :

$$N!/((N-n)! * (n)!)$$

5.1.3 40 assets market

Now take a look at the 40 assets market.
Once more, we see (in Figures 5.7 & 5.8) that for a small number of changes, the

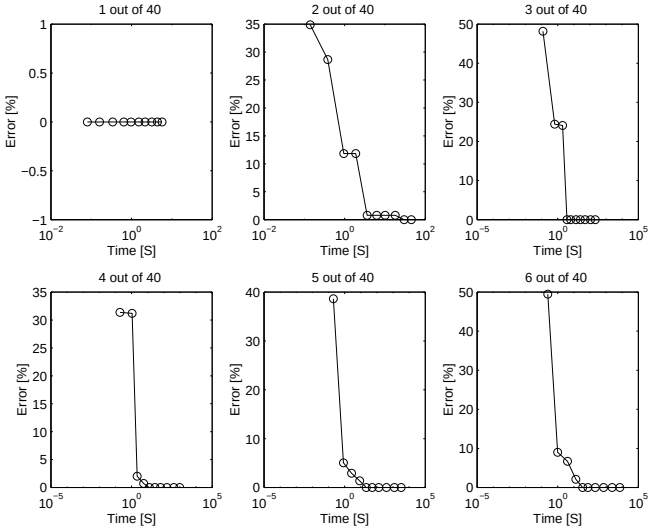

Figure 5.7: Results with the 40 assets market, for 1 to 6 changes.

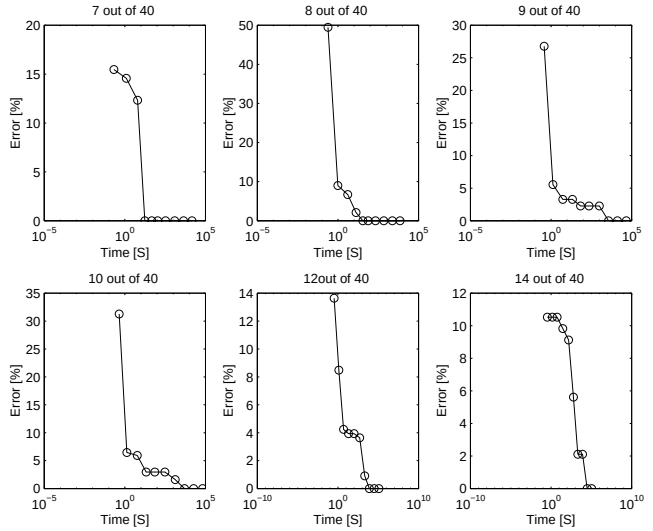

Figure 5.8: Results with the 40 assets market, for 7 to 10, 12 and 14 changes.

convergence is fast. Moreover, it seems that the convergence is even faster than the one we see in the preceedings markets. In fact, for $n = 1$ to 10, after the fifth iteration we already have an error smaller than 5%. Even more, this results are found in 100 seconds, that means in less than 2 minutes. The comportement is almost the same than for the preceedings markets for greater numbers of changes (see Figures 5.8 & 5.9).

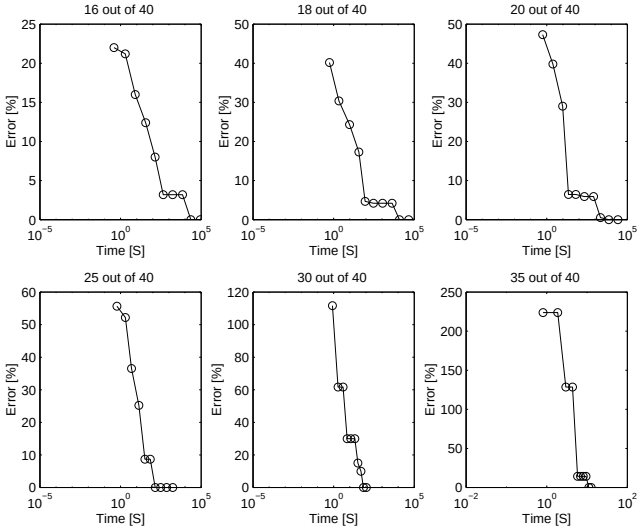

Figure 5.9: Results with the 40 assets market, for 16, 18, 20, 25, 30, 35 changes.

5.1.4 50 assets market

For the 50 assets market, we just run, for time reasons, the algorithm for 1 to 9 changes allowed. In Figure 5.10, our constatation holds again. Moreover, it is

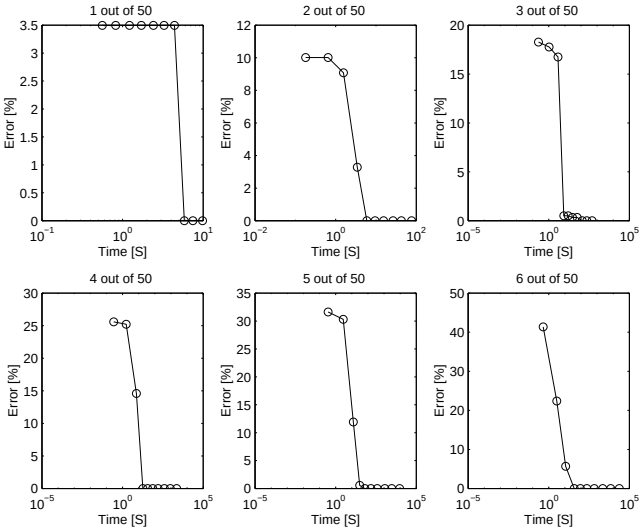

Figure 5.10: Results with the 50 assets market, for 1 to 6 changes.

ones again better. After five iterations, we are at less than 1% of the optimum, and this takes about 100 seconds again.

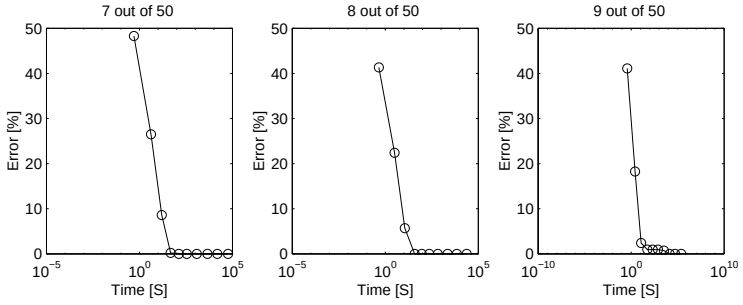

Figure 5.11: Results with the 50 assets market, for 6 to 9 changes.

5.2 Test of dependency on the market

All the preceeding was done with the same market. In this section we want to test the dependency on the market, i.e. we want to test if we where lucky in choosing our markets, or if it does not depends on which market we take.

To test that, we took 20 markets at random, each of them with 30 assets. Then we run the algorithm with $n = 7$.

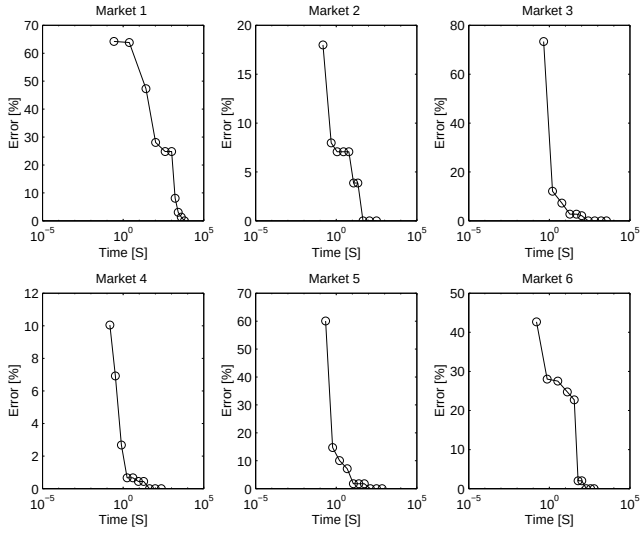

Figure 5.12: Results for 7 changes in random markets 1 to 6

In Figures 5.12, 5.13, 5.14 and 5.15, we see that most of the time the convergence is good, that means after the fifth iteration, thirteen of the error are smaller than ten percent. After the sixth iteration, eighteen of them are smaller than ten percent. But in these eighteen, fifteen are smaller than five percent. After the seventh iteration, every error is less than ten percent.

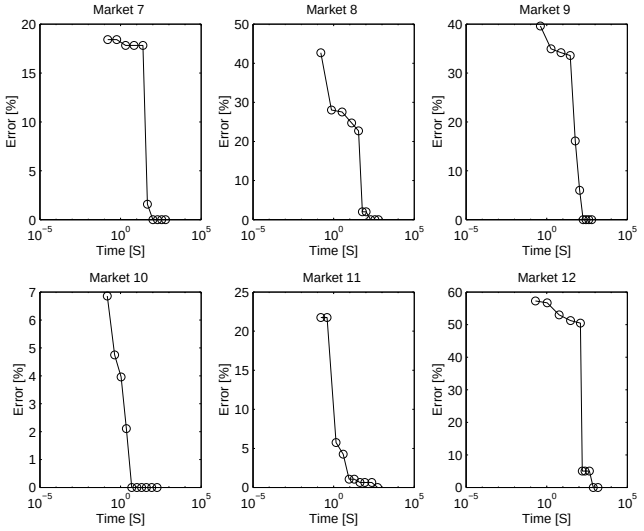

Figure 5.13: Results for 7 changes in random markets 7 to 12

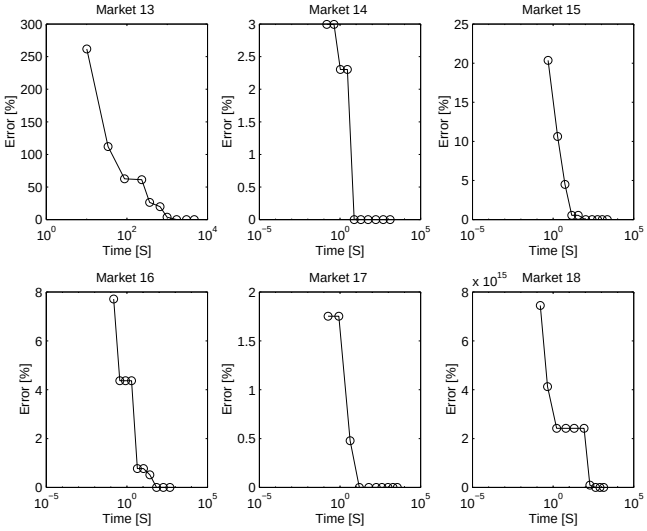

Figure 5.14: Results for 7 changes in random markets 13 to 18

We must also take into account that these calculations are done with 30 assets markets. As we see before (5.1.1 to 5.1.4) there seems to be a tendency for the convergence to be faster as the size of the market grows.

Let us recall that after the i^{th} iteration, we are sure to have a solution which is in $[y^*, y^* * 10/i]$, where y^* is the value of the exact optimal solution. That means that after the seventh iteration we have a potential error of $100 * |1 - 10/7|\%$, which is about 42%. Thus we can only say that after the seventh iteration we are sure to have an error smaller than 43%, but there is a great probability to have an

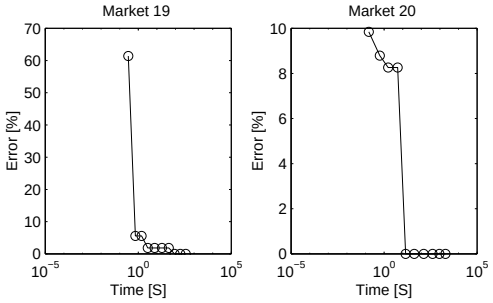

Figure 5.15: Results for 7 changes in random markets 19 and 20

error smaller than 10%. However, this result is quite strong and allows us to say that the method can be usefull, even if we could perhaps find a counter-exemple, where the error is maximal.

5.3 Test with bigger markets

5.3.1 100 assets market

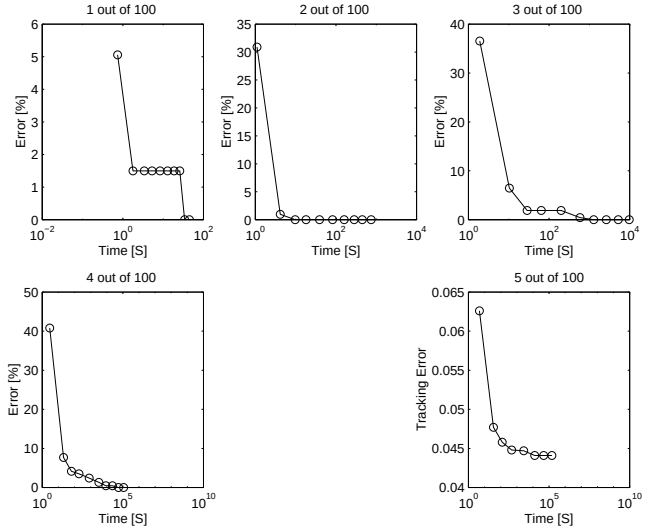

Figure 5.16: Results for a 100 assets market.

In Figure 5.16, we see that after two iterations we already are under ten percent for $n = 1$ to 4. For five changes, even if we did not let the algorithm run till the end, we can hope for a convergence.

5.3.2 200 & 500 assets market

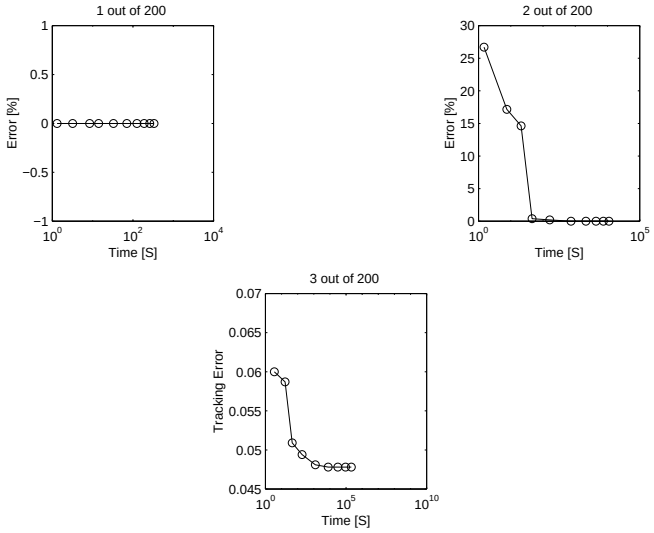

Figure 5.17: Results for a 200 assets market.

For the 200 assets market (Figure 5.17), we see that this convergence is also fast.

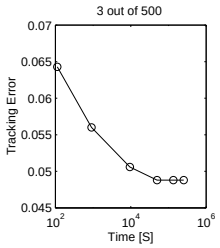

Figure 5.18: Results for a 500 assets market.

In Figure 5.18, we see that the process seems to converge also quite quickly for the 500 assets market. It seems that we get a good approched solution in a reasonable time (10^4 seconds).

5.4 Other considerations on the results

Another idea was to compare how does the tracking error decrease as we increase the number of assets we are allowed to modify. The goal was to see if the tracking error had a second derivative which was positive. If it was so, we could find a number of changes after which it was useless to go, because the cost for one more change was bigger than the risk we gain.
Since we've calculated the optimum for a wide range of change for the 20, 30 and 40 assets market, we can plot the sensibility of the tracking error to the number of changes.

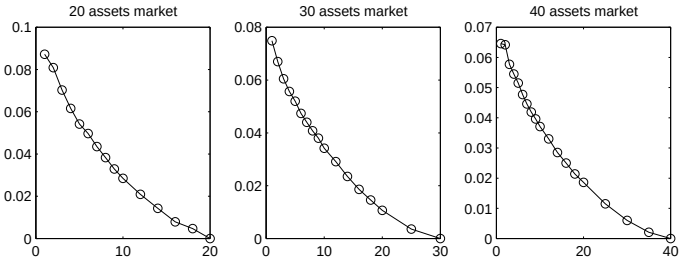

Figure 5.19: Tracking error against the number of changes allowed.

In figure 5.19, we see there is no evidence for the strongly positivity of the second derivative of the tracking error against the number of changes. We thus can not give a kind of upper bound for the number of changes.

Conclusion

In conclusion, we can not say that our goal was completely reached, since we did not find an algorithm faster to hit the optimum. Nevertheless, our goal was partially fullfield since the method we find is relatively fast to get a good approximation of the optimum. What we see in chapter 5 is that our algorithm is converging in many case, but we can not give a better theoretical insurance on this convergence than the one given in chapter 4, i.e. the lower bound for the optimum is after step i equals : $|1 - 10/i| *$ solution after step i.

Moreover, we remark that our algorithm is converging faster and better when the number of changes allowed is small (about 20% of the total number of assets). As we mentionned in chapter 5, the convergence seems also to be faster as we increase the size of the market. All that makes us think that this method could be usefull in practice.

The further developments of this study could be to study several maner of decreasing the "percent". Another one could be to find a way to calculate a better lower bound for the subtrees. Let us mention also that we can gain a litle time in compiling the matlab code in C, but that would not change fundamentaly our results.

Appendix A

Matlab Listing

A.1 Main procedure

```
function [xmin,ymin,wmin]=evolutif(V,n,wB,wS,evalf,IRs,fs)

% [xmin,ymin,wmin]=evolutif(V,n,wB,wS,evalf,IRs,fs)
%
% Main procedure to calculate approximation and the optimal solution
%
% Parameters
% V: covariance matrix
% n: number of trades that are allowed
% wB: benchmark
% wS: optimal active strategy portfolio
% evalf: m-file, interface algorithm-problem (see evalte.m)
% IRs: information ratio of portfolio S
% fs: upper bound for the branch & bound

flopsstart=flops;   %initialization of the number of Matlab operations
clockstart=cputime; %initialization of CPU time
V1=V;               %Backup of the initial value
wB1=wB;
wS1=wS;
IRs1=IRs;
fsinit=fs;
for i=1:10,        %ith step
  percent=i/10     %definition of "percent"
  [xmin,ymin,wmin]=bab(V,n,wB,wS,evalf,IRs,fs,percent) %"percented" B&B
  if isempty(xmin),    %test to see if there was no admissible solution
    V=V1;
```

```
    wS=wS1;
    wB=wB1;
    IRs=IRs1;
  else,                 %reinitialization of the inputs data
    fs=ymin;
    V=V1;
    wS=wS1;
    wB=wB1;
    IRs=IRs1;
  end;
end;
flopstotal=flops-flopsstart %total number of Matlab operations
clocktotal=cputime-clockstart %total cpu time
```

A.2 Branch & Bound procedure

```
function [xmin,ymin,wmin]=bab(V,n,wB,wS,evalf,IRs,fs,percent)

% [xmin,ymin,wmin]=bab(V,n,wB,wS,evalf,IRs,fs,percent)
%
% "percented" branch and bound method to calculate an approched solution
%
% Parameters
% V: covariance matrix
% n: number of trades that are allowed
% wB: benchmark
% wS: optimal active strategy portfolio
% evalf: m-file, interface algorithm-problem (see evalte.m)
% IRs: information ratio of portfolio S
% fs: upper bound for the branch & bound
% percent: coefficient of the lower bound


global  xmin ymin wmin fsup lbound lbback k
fsup=fs;       % initialisation of the upper bound
lbound=[];     % initialisation of the lower bound
k=0;           % initialisation of the number of leaves evaluated
fstart=flops;
cstart=cputime;
beta=V*wB'/(wB*V*wB');        % calculation of the preference vector
```

```
wSR=wS-wS*beta.*wB;
omegaS=sqrt(wSR*V*wSR');
alpha=IRs/omegaS.*V*wSR';
order=abs(alpha).*abs(beta-1).*abs(wS'-wB');
P=permu(order);               % permutation matrix
V=P'*V*P;                     % permutation of V, wB and wS
wB=wB*P;
wS=wS*P;


left([],V,n,wB,wS,IRs,evalf,percent); % recursive call of the root
xmin=xmin*P';        % final permutation to resume the initial order
wmin=wmin*P';

fend=flops-fstart            %Number of Matlab operations used
cend=cputime-cstart          %CPU time
k                            %Number of leaves evaluated
```

A.3 Evaluation procedure

```
function [w,y]=evalte(x,N,V,wB,wS,n,IRs)

% [w,y]=evalte(x,N,V,wB,wS,n,IRs)
%
% Parameters
% x: portfolio
% N: Market size
% V: covariance matrix
% wB: benchmark
% wS: optimal active strategy portfolio
% n: number of trad allowed
% IRs: information ratio of portfolio S

beta=V*wB'/(wB*V*wB');  %calculation of the beta
e=ones(length(wB),1);

Nb=length(x);
id=1:Nb;
xrt=x.*id;
nxrt=~x.*id;
xr=[];                %in xr: index of non nul elements of x
```

```
nxr=[];                %in nxr: index of nul elements of x
for i=1:length(xrt),  %calculation of xr and nxr
   if xrt(i)~=0,
     xr=[xr,xrt(i)];
     else
      nxr=[nxr,nxrt(i)];
   end
end


iVx=inv(V(xr,xr));   %decomposition of the inputs according to xr and nxr
ex=e(xr);
betax=beta(xr);


if (sum(x)==N)
   wBnx=0;
   Vnxx=0;
   betanx=0;
   enx=0;
else
   wBnx=wB(nxr);
   Vnxx=V(nxr,xr);
   betanx=beta(nxr);
   enx=e(nxr);
end

a1=betax'*iVx*betax;   %Lagrangian optimization
a2=ex'*iVx*betax;
b1=betax'*iVx*ex;
b2=ex'*iVx*ex;

k1=Vnxx'*wBnx'-V(xr,:)*wS';
c1=wS*beta-wBnx*betanx+betax'*iVx*k1;
c2=1-wBnx*enx+ex'*iVx*k1;

lambda1=(c1*b2-b1*c2)/(a1*b2-b1*a2);
lambda2=(c2*a1-a2*c1)/(a1*b2-b1*a2);

wx=iVx*(lambda1*betax+lambda2*ex-k1);
w=wB;
w(xr)=wx;
```

```
y=sqrt((w-wS)*V*(w-wS)'); %calculation of the objective function
```

A.4 Stop procedure

```
function stop(fix,b,V,n,wB,wS,IRs,evalf)

% stop evaluates the feasible solutions found
% and updates the best actual solution and the
% upper bound

global xmin ymin wmin fsup
N=length(wB);
% construction of the feasible solution
fix(length(fix)+1:N)=ones(1,N-length(fix)).*b;

[w,y]=feval(evalf,fix,N,V,wB,wS,n,IRs); % evaluation

if y<=fsup      % this solution is better than the actual best one
   ymin=y;
   xmin=fix;
   wmin=w;
   fsup=y;
end
```

A.5 Left recursive procedure

```
function left(fix,V,n,wB,wS,IRs,evalf,percent)

% recursive function treating the left-hand son
% of the node specified by the vector fix

global fsup lbound lbback k

N=length(wB);

if (sum(fix)==n)  % the resting positions must be zeros
   stop(fix,0,V,n,wB,wS,IRs,evalf);
   k=k+1;
elseif (sum(~fix)==N-n) % the resting positions must be ones
```

```
   stop(fix,1,V,n,wB,wS,IRs,evalf);
   k=k+1;
else
   [w,y]=feval(evalf,[fix,ones(1,N-length(fix))],N,V,wB,wS,n,IRs);
       % evaluation to determine the lower bound of the node
   if y<fsup*percent % the node is explored only if its lower bound is
             %  smaller than the actual upper bound times percent
      if isempty(lbound) % lbound is not defined if there is no node
                    % with only one son executed
         lbound=y;  % in that case the lbound corresponds to
                    % the actual node
      end
      right([fix,1],V,n,wB,wS,IRs,evalf,percent);% recursive function call
      if lbound==y       % we will explore the left-hand son of the
                         % node specifying the actuel lower bound
         lbback=lbound;  % backup of the lower bound
         lbound=[];      % there will be a new lower bound
      end
      left([fix,0],V,n,wB,wS,IRs,evalf,percent);% recursive function call
   end
end
```

A.6 Right recursive procedure

```
function right(fix,V,n,wB,wS,IRs,evalf,percent)

% recursive function treating the right-hand son
% of the node specified by the vector fix

global fsup k
N=length(wB);
if (sum(fix)==n)        % the resting positions must be ones
   stop(fix,0,V,n,wB,wS,IRs,evalf);% evaluation of the feasible sol.
   k=k+1;
elseif (sum(~fix)==N-n) % the resting positions must be zeros
   stop(fix,1,V,n,wB,wS,IRs,evalf); % evaluation of the feasible sol.
   k=k+1;
else
   right([fix,1],V,n,wB,wS,IRs,evalf,percent); % recursive function call
   left([fix,0],V,n,wB,wS,IRs,evalf,percent);  % recursive function call
end
```

A.7 Permutation procedure

```
function P=permu(a)
% P=permu(a)

N=length(a);
sorta=sort(abs(a));  %sort by absolute value
sorta=sorta(N:-1:1); %reverse the vector

per=zeros(1,N);

for i=1:N    %calculation of the permuted matrix
   pos=find(abs(a(i))==sorta);
   k=1;
   while sum(pos(k)==per)
      k=k+1;
   end
   per(i)=pos(k);
end

P=zeros(N,N); %Calculation of the permutation matrix
for i=1:N
   P(i,per(i))=1;
end
```

A.8 Technical procedure

```
function [w,y]=feval(evalf,x,N,V,wB,wS,n,IRs)

%This function is included in some version of Matlab. It evaluate the
%function "evalf" with the resting parameters as inputs

evalf(x,N,V,wB,wS,n,IRs);
```