Mean-Variance Portfolio Selection With Complex Constraints

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)

von der Fakultät für
Wirtschaftswissenschaften
der Universität Karlsruhe (TH)

genehmigte

DISSERTATION

von
Dipl.-Wi.-Ing. Michael Stein

Tag der mündlichen Prüfung: 06.06.2007

Referent: Prof. Dr. Hartmut Schmeck
Korreferentin: Prof. Dr. Marliese Uhrig-Homburg

Karlsruhe, 2007

Acknowledgements

The main parts of the research documented in this thesis were conducted during the years I worked at the Institute of Applied Informatics and Formal Description Methods (AIFB) in Karlsruhe. The productive and nevertheless relaxed working climate there and the helpfulness of my co-workers deserves to be highly praised.

Of the many persons that contributed – directly or indirectly – to this work, I would especially like to thank my thesis advisor Prof. Dr. Hartmut Schmeck for his unwavering support and for the freedom to always follow my own ideas. I also want to express my gratitude to Prof. Dr. Marliese Uhrig-Homburg and to Prof. Dr. Oliver Stein for reviewing my thesis.
Particularly, I want to thank Dr. Jürgen Branke for many productive, interesting and encouraging discussions over the past few years. The joint work with him provided the foundation for what is now the final "product" of my efforts.

I would also like to mention that the research for this thesis was financially supported by the Schleicher Foundation, to which I hereby want to express my sincere gratitude.

Finally, I would like to thank my family for their permanent support and for providing me with a "safe haven" even in stressful and chaotic times.

Karlsruhe, September 2007 *Michael Stein*

Contents

List of Figures

List of Tables

1 Introduction

In the 1950s, when an important part of the theoretical basis for what is now known as mean-variance portfolio selection was developed, few would have imagined what kind of computational processing power would be available half a century later. This development makes it possible to solve extensions of the original portfolio selection problem that were considered to be intractable at that time – if they were considered at all.
The initial idea for this thesis is based on the following observation: it is impossible to transform the regulatory framework mutual funds have to comply with in Germany into constraints that are easily integrated into classical mean-variance optimization. With the large amount of money invested into these financial instruments, it does seem reasonable to take a closer look at this problem in order to analyze what extensions to the classical optimization model are required and how they influence the problem solving methodology.
Other constraints that do not fit into the "mold" of classical mean-variance optimization are treated in several publications, in particular the so called *cardinality constraints*, *buy-in thresholds*, and *integer constraints*. Of these three types only cardinality constraints and buy-in thresholds are relevant for large-scale investment, which is the reason we will focus on these categories and not on integer constraints. When looking at the published approaches capable of integrating these nontrivial constraints into the optimization, one cannot fail to notice that they mainly rely on metaheuristics and often seem to be not especially well adapted to the task. We have found that cardinality constraints and buy-in thresholds can be handled in a similar fashion as the "difficult" constraints that are based on the German investment law. Due to the fact that only those two categories are relevant for large scale investment, the topic of this thesis has been extended to additionally include cardinality constraints and buy-in thresholds.
Algorithms for portfolio optimization may have different objectives. If the mean-variance paradigm is accepted and no special investor preference is given, then the "perfect" algorithm would compute the set of Pareto optimal solutions in an immeasurably short time for all problem sizes and all types of constraints. As there is no algorithm with this capability, the goal can only be to get as close as possible to this "perfect" algorithm. The two main objectives in the design of the algorithm are therefore speed and solution quality. The main goal of this thesis was to develop different types of algorithms that can cope with nonconvex constraints and that are either able to produce a solution of sufficient quality very quickly, or that are able to compute a very good solution, but need more time. The design of these algorithms can therefore be interpreted as a new bicriteria optimization problem that has to be solved.

This thesis is structured as follows:
In Chapter 2, a short rationale for the economic model and the resulting problem formulation – the mean-variance model – is given. The drawbacks of the model are highlighted, and we briefly touch upon the usage of different risk measures like e.g. Value at Risk[1]. We also introduce different types of constraints, namely those that can be easily integrated into the standard portfolio selection algorithms (linear constraints), and those that can not (nonconvex constraints).
Chapter 3 aims at providing a fast and reliable algorithm that is capable of calculating the complete Pareto front for the mean-variance portfolio selection problem when there are only linear equations and inequalities, i.e., when the feasible region defined by the constraints is convex. Different algorithm variants for this problem are examined, and we describe how to implement them in an efficient manner. Due to several algorithmic improvements, the fastest implementation is capable of calculating the complete Pareto front in a very short amount of time even on large problem instances.
In Chapter 4, we describe several heuristics that belong to the category of so called ϵ-Constraint methods and are able to approximate the Pareto front for problems with nonconvex constraints like e.g. cardinality constraints. This is accomplished by transforming the expected return criterion into an additional linear inequality. Two aspects of the ϵ-Constraint algorithms are discussed, namely how to best calculate a single point and how to distribute these points along the Pareto front. The developed algorithm variants are then evaluated in a series of extensive tests.
The next chapter, Chapter 5, describes a new algorithm, the envelope-based multi-objective evolutionary algorithm, developed especially for mean-variance portfolio selection with nonconvex constraints. Instead of typical evolutionary algorithms in which an individual represents a single portfolio, in our algorithm, an individual corresponds to a set of convex constraints. In a series of tests, we compare this new type of algorithm with a state-of-the-art multi-objective evolutionary algorithm and are able to show that our algorithm is superior with respect to convergence rate, solution quality during the algorithm run, and final solution quality.
In the last major chapter we propose to integrate algorithm parts from Chapters 4 and 5 in order to create a faster and even better algorithm for portfolio selection problems with nonconvex constraints. The comparison with the previous results demonstrates that the newly designed procedures are able to compute better results in less time than the purely point-based approaches.
We conclude the thesis with a brief summary and outlook in Chapter 7. Several additional diagrams with test results can be found in the Appendix.

[1] In the remainder of the thesis, we assume the validity of the mean-variance model, and therefore only expected return (mean) and variance are used as optimization criteria.

2 The Mean-Variance Optimization Model

In the first section of this chapter, we will briefly summarize the economic principles that form the basis for mean-variance portfolio selection. An introduction similar to the one presented here can be found in any of the standard references for financial economics (see e.g. Huang and Litzenberger [HL88], or LeRoy and Werner [LW01]) or in the books from Markowitz [Mar59, Mar87].
Section 2.2 introduces the mean-variance model for portfolio selection. It consists of two optimization criteria (variance and expected return) and an arbitrary number of linear equations and inequalities. Three different approaches how to compute a solution for this model are presented: the ϵ-Constraint approach, the weighting method, and parametric quadratic programming. In Section 2.3 we describe other possible dispersion measures besides the variance which better capture the notion of risk, and Section 2.4 explains the origin of the benchmark problems we will use for testing in the remainder of the thesis. In Section 2.5, which concludes this chapter, we specify several types of constraints that may be relevant for portfolio selection problems, we categorize them and analyze their influence on the difficulty of the optimization process.

2.1 Fundamentals of Portfolio Selection

In a market economy, nearly everybody regularly has to solve a variation of the problem that lies at the core of portfolio selection: what to do with a given amount of money in order to get the highest degree of overall well-being. This problem description is very vague. In order to handle it quantitatively, several additional assumptions, simplifications, and formalizations have to be made.
In economics, "well-being" is often measured with the help of a *utility function* $u : Y \mapsto \mathbb{R}$, that maps every possible outcome Y for an event to a real number. A higher objective function value indicates a higher degree of well-being.
The first assumption we make – which is rather general – is that the investor is only interested in financial gain. Other motivations, like e.g. the preference of investments that are ethically unobjectionable, are not considered.
Another important simplification is the assumption that the investment process can be expressed as a so called *one-period model*. In a one-period model, the investment decision is taken at a point in time t_0, and during the period $\triangle t$ the decision is not or can not be revised. At $t_1 = t_0 + \triangle t$, each investment offers a specific yield. The investor's goal in this model is to maximize his end-of-period wealth W_1. What makes this decision problem nontrivial is that for some or all investments the end-of-period yield is not known in t_0, which makes the problem non-deterministic.
One-period models certainly have serious drawbacks, as it is hardly imaginable that an investor will stand by and do nothing if she receives important information during the

period $\triangle t$ that would cause her to adapt her investment positions to the new circumstances. Unfortunately, more advanced models that allow multiperiod transactions or even continuous buying and selling introduce a degree of complexity that is not easy to handle. They also require either additional information, e.g. about the consumption preferences of the investor, or they make general assumptions in that direction (e.g. only the terminal wealth is of interest). As the algorithms allowing the integration of complex constraints are our main topic, an additional treatment of multiperiod or continuous models would by far exceed the scope of this thesis. Therefore, for an introduction and an overview of the different methodologies applied in multiperiod portfolio selection, the reader is referred to Steinbach [Ste01]. A good starting point for continuous models in general is a survey by Sundaresan [Sun00].
We assume further that the investment decision in the presence of uncertainty is based on the so-called *expected utility hypothesis*, which says that the optimal decision under uncertainty is the one that maximizes expected utility (cf. von Neumann and Morgenstern [vNM44]). The expected utility hypothesis is not without contentious points, as documented e.g. by Ellsberg [Ell61], Kahnemann and Tversky [KT79, KT04], or Rabin [Rab00]. Nevertheless, the hypothesis is accepted in many standard texts and will be presumed to be valid in the remainder of this thesis.
The Taylor-Expansion of the utility function $u(W_1)$ at position $E(W_1)$ results in the following equation:

$$\begin{aligned} u(W_1) = u(E(W_1)) + u'(E(W_1))\,(W_1 - E(W_1)) \\ + \frac{1}{2}u''(E(W_1))(W_1 - E(W_1))^2 + \sum_{n=3}^{\infty} \frac{1}{n!} u^{(n)}(E(W_1))m^n(W_1) \end{aligned} \tag{2.1}$$

where $E(W_1)$ is the expected end-of-period wealth and $m^n(W_1)$ is the nth moment of W_1 at position $E(W_1)$.
When Eq. 2.1 is used to express the expected utility of the investor, we get the following result:

$$\begin{aligned} E(u(W_1)) &= E(u(E(W_1))) + E(u'(E(W_1))(W_1 - E(W_1))) \\ &\quad + E(\frac{1}{2}u''(E(W_1))(W_1 - E(W_1))^2) + E(\sum_{n=3}^{\infty} \frac{1}{n!} u^{(n)}(E(W_1))m^n(W_1)) \\ &= u(E(W_1)) + u'(E(W_1))E(W_1 - E(W_1)) \\ &\quad + \frac{1}{2}u''(E(W_1))E((W_1 - E(W_1))^2) + E(\sum_{n=3}^{\infty} \frac{1}{n!} u^{(n)}(E(W_1))m^n(W_1)) \\ &= u(E(W_1)) + u'(E(W_1))(E(W_1) - E(W_1)) + \frac{1}{2}u''(E(W_1))V(W_1) \\ &\quad + E(\sum_{n=3}^{\infty} \frac{1}{n!} u^{(n)}(E(W_1))m^n(W_1)) \\ &= u(E(W_1)) + \frac{1}{2}u''(E(W_1))V(W_1) + \underbrace{E(\sum_{n=3}^{\infty} \frac{1}{n!} u^{(n)}(E(W_1))m^n(W_1))}_{s} \end{aligned} \tag{2.2}$$

The main argument of Markowitz [Mar59, Mar87] is that if the utility function of the investor is quadratic or if it can be approximated with sufficient precision by a quadratic function, then the term s in Eq. 2.2 becomes 0. In this case, expected utility can be expressed solely in terms of expected return $E(W_1)$ and variance $V(W_1)$. If it is further assumed that the utility function is concave, i.e. the second derivative is negative, then from all portfolios with the same expected return the one with the smallest variance maximizes expected utility.
Concave utility functions that are quadratic have one serious drawback: For each one there is an input value above which the gradient of the utility function becomes negative. An increase in wealth would therefore decrease utility, which is not compatible with what would usually be seen as a rational behavior.
Markowitz [Mar59, Mar87] argues that for reasonable utility functions the quadratic approximation should be good enough in a sufficiently large area around $E(W_1)$ to prevent a major loss caused by approximation errors. Empirical results by Kallberg and Ziemba [KZ83] and Kroll et al. [KLM84] confirm this proposition.

If the utility function is not quadratic, there is a second reason why it makes sense to focus solely on expected return and variance. Under the condition of multivariate normally distributed asset returns, the return distribution of every portfolio consisting of those assets is also Gaussian, due to the fact that the normal distribution is stable. Moreover, any normal distribution is completely defined by its first and second moment (i.e. expected return and variance). Therefore, as long as the investor is risk averse, the conclusion is the same as above, irrespective of the type of utility function: for any given value of expected return, the portfolio with the smallest variance maximizes expected utility.
One obvious problem with using Gaussian distributions to model asset returns is based on the attribute of the normal distribution to be unbounded from below: if the investment alternatives are regular shares, their value can not fall below 0, i.e. there is not even the smallest probability for the return to be smaller than -1.
The main argument against normality of the asset returns is, however, that there is a lot of empirical evidence that investment returns are not multivariate Gaussian. Classical references documenting this are e.g. Mandelbrot [Man63] and Fama [Fam65].
But even if neither of the circumstances mentioned above (quadratic utility or normally distributed asset returns) are assumed to be true, there is a good chance that the portfolio that maximizes expected utility is fairly close to the one that minimizes variance for a given value of expected return (see e.g. Kroll et al. [KLM84] or Cremers et al. [CKP03]).
In the problematic case that expected utility has to be maximized with neither quadratic utility nor normally distributed returns, there is no other choice but to explicitly determine the utility function of the investor – which is often not an easy task – and then to directly use it in the optimization process. Depending on the type of the utility function, this may be nearly impossible to do in a reasonable amount of time.
For this reason, this thesis is restricted to mean-variance optimization.

2.2 The Standard Mean-Variance Model

The standard one-period mean-variance (MV) optimization problem can be expressed as a bicriteria optimization model where the solution simultaneously maximizes expected return and minimizes portfolio variance with respect to a given set of equality and inequality constraints:

Standard Mean-Variance Model (SMVM)

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} \tag{2.3a}$$

$$\max E(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\mu} \tag{2.3b}$$

subject to

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.3c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{2.3d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.3e}$$

Element x_i of the vector $\boldsymbol{x} \in \mathbb{R}^n$ denotes the fraction of the budget invested in asset i. $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ is the covariance matrix, $\boldsymbol{e} \in \mathbb{R}^n$ represents the unit vector, $\boldsymbol{\mu}$ is the vector of expected returns of all assets. $\boldsymbol{A_I}$ and $\boldsymbol{A_E}$ are the coefficients matrices of inequalities and equalities; $\boldsymbol{b_I}$ and $\boldsymbol{b_E}$ denote the corresponding right hand sides. Equation 2.3d (the budget constraint) guarantees that the fractions of the budget add up to 1. The budget constraint can be easily expressed as a part of the equations that are modeled by $\boldsymbol{A_E x} = \boldsymbol{b_E}$. We have mentioned it separately, however, as the constraint is often written down explicitly in other publications as well, probably due to its effect to normalize the solutions.

If the investor does not have to spend the complete budget, i.e., if he is allowed to keep a cash reserve (or if he can invest in a riskless asset), this can be easily integrated into the model by adding an asset with the desired yield (0 or a riskless interest rate) and a standard deviation of 0. Additionally, the "new" asset has to be uncorrelated to all other assets[1].

Other types of constraints compatible with the standard model but often mentioned separately are e.g. the prohibition of short sales, sector constraints, and upper bounds on asset weights. They are discussed in more detail in Section 2.5.

The necessary data for the mean-variance model consists of the expected return for every asset – an n-vector – and the respective $n \times n$ covariance matrix. Since the covariance matrix is symmetric, we require in total n variances and $n(n-1)/2$ covariances. In total we therefore need to acquire $2n + n(n-1)/2 = \frac{1}{2}n(n+3)$ data elements prior to the actual mean-variance optimization. To find a good estimate for that many numbers is of critical importance, since even small estimation errors can have grave consequences for results of the optimization. Kallberg and Ziemba [KZ83] and Chopra and Ziemba [CZ93] have found that mean-variance optimization is especially sensitive to variations of the

[1] If we assume that the riskless asset has the index k, this is achieved by setting all elements of the covariance matrix that have either row or column index k to 0.

expected returns. Best and Grauer [BG91] described the analytical framework to perform sensitivity analysis with respect to changes of the vector of expected returns and the right hand sides of the constraints. This framework is closely related to parametric quadratic programming algorithms that are discussed extensively in Chapter 3. A more general discussion of the difficulties to apply mean-variance analysis in practice can be found in Michaud [Mic89], although again the main focus is put on the sensitivity of the input data.
Several publications propose techniques to get better estimates of both expected returns and the covariance matrix. See e.g. Jobson and Korkie [JK80], Black and Litterman [BL92], Chopra et al. [CHT93], Ledoit and Wolf [LW04], Elton et al. [EGS06] and the references therein. Following a different approach, Jagannathan and Ma [JM03] propose to introduce nonnegativity constraints instead of more advanced parameter estimation techniques.
This thesis is concerned mainly with the actual optimization algorithms and not with the generation of the required input data. In the remainder of the thesis we therefore assume that the given data (the vector of expected returns and the covariance matrix) is correct.
There is usually no single portfolio that both minimizes variance and maximizes expected return. Instead, the result of an optimization based on the SMVM is generally a set of efficient portfolios.

Definition. A portfolio is *efficient / Pareto optimal* in the context of mean-variance portfolio selection if and only if there is no other feasible portfolio that improves at least one of the two optimization criteria without worsening the other.

When a portfolio is efficient, there is no other portfolio that complies with the constraints and has

1. lower variance and higher expected return or
2. lower variance and the same expected return or
3. the same variance and higher expected return.

The set of all efficient portfolios is called the *Pareto front*, *Pareto Frontier*, or the *Efficient Frontier*.

There are three well-established approaches to calculate a "solution" for problem SMVM: the ϵ-Constraint approach, the weighted sum method, and algorithms for parametric quadratic programming. Which of these is to be selected depends on the goal of the optimization, and on the capabilities of the software packages that are available for the task.
We will briefly discuss all three in this section[2].

[2] An extended presentation of the parametric quadratic programming approach can be found in Chapter 3, and the ϵ-Constraint method plays an important role in Chapter 4.

ϵ-Constraint Approach

If it is our intention to find the point on the Efficient Frontier with the minimum variance that has an expected return of at least E_f, this automatically removes one objective function and introduces an additional constraint. The resulting optimization problem is – as the covariance matrix is positive semidefinite – a convex quadratic programming problem (QP):

ϵ-Constrained Quadratic Programming Model (ϵ-QPM)

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} \tag{2.4a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{\mu} \geq E_f \tag{2.4b}$$

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.4c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{2.4d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.4e}$$

The solution of this model can be easily computed by using a QP-solver from one of several more advanced optimization software packages. A list of suitable programs and libraries is provided by the NEOS Guide [NEO06].
Such a solution, however, represents only one point on the Efficient Frontier. An *approximation* of the complete Pareto front can be computed by repeatedly solving the ϵ-QPM with increasing (or decreasing) E_f. In multicriteria optimization, this methodology is usually called ϵ-*Constraint method*. For further information on the ϵ-Constraint approach from a general multicriteria point of view, the reader is referred to Changkong and Haimes [CH83], or to Miettinen [Mie98].
One main drawback attributed to the ϵ-Constraint method is the time it requires to generate a sufficiently precise approximation, as the ϵ-QPM has to be solved for a large number of different values of E_f. Steuer et al. [SQH06] measured the time it took for only a very crude approximation (20 different values of E_f) with a commercial optimization package. Their conclusion was that for larger problem sizes, the slowness of the approach made this method inferior to parametric quadratic programming.
Unfortunately, the ϵ-Constraint method is the only approach most software packages and toolboxes offer for portfolio selection (for more details, see Steuer et al. [SQH06]).

Weighting Approach

The weighting method is another very basic but widely used approach in multicriteria optimization (Miettinen [Mie98]). In the field of portfolio selection, models of this type are regularly employed. Furthermore they form the basis for parametric quadratic programming algorithms. By default, a model based on the weighting methodology is similar to the ϵ-QPM insofar as its solution is also just a single point on the Pareto front.

Instead of turning one objective into an additional constraint, however, the "new" single objective function F is a weighted sum (or difference) of both objective functions from the SMVM:

λ_e-QPM

$$\min F(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} - \lambda_e \boldsymbol{x}^T \boldsymbol{\mu} \tag{2.5a}$$

subject to

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.5b}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{2.5c}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.5d}$$

In order to approximate the Efficient Frontier, the λ_e-QPM has to be solved for different values of λ_e. It is sufficient to look at the solutions for $\lambda_e \geq 0$, as with them, all the points on the Efficient frontier can be calculated. The same procedures that solve the ϵ-QPM can be used here as well, as most quadratic programming solvers permit a linear term in the otherwise quadratic objective function.
The solution sets that can be calculated for varying parameters (either E_f or λ_e) are the same for both models: The Lagrange functions are identical if the parameter λ_e from model λ_e-QPM is interpreted as the multiplier for the expected return constraint in the ϵ-QPM[3]:

$$L(\boldsymbol{x}, \boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\lambda_e}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} - \lambda_e \, \boldsymbol{x}^T \boldsymbol{\mu} + \boldsymbol{\nu}^T (\boldsymbol{A_I x} - \boldsymbol{b_I}) + \boldsymbol{\lambda}^T (\boldsymbol{A_E x} - \boldsymbol{b_E})$$

As a consequence, the Karush-Kuhn-Tucker conditions – which are necessary and sufficient for optimality if the objective function and the constraints are convex – are identical as well:

$$\nabla L(\boldsymbol{x}, \boldsymbol{\nu}, \boldsymbol{\lambda}, \boldsymbol{\lambda_e}) = 0 \tag{2.6a}$$

$$\nu_i \, (\sum_{j=1}^{N} a_{ij} x_j - b_i) = 0 \qquad \forall \, i = 1, \ldots, l \tag{2.6b}$$

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.6c}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.6d}$$

$$\boldsymbol{\nu}, \lambda_e \geq 0 \tag{2.6e}$$

Each value of $\lambda_e \in [0; \infty)$ is mapped to exactly one value of E_f. For a proof and further details, the reader is referred to Markowitz [Mar87]. It is obvious that for $\lambda_e = 0$ the solution of the λ_e-QPM is the *Minimum Variance Portfolio (MVP)*, and that if λ_e is large enough, the calculated solution is the portfolio with maximum expected return.

[3] For sake of brevity, the budget constraint $\boldsymbol{x}^T \boldsymbol{e} = 1$ has been considered as part of the general equations.

Parametric Quadratic Programming Approach

If the Pareto front as a whole has to be calculated for a portfolio selection problem of type SMVM, the only choice is an active set algorithm for parametric quadratic programming (cf. Chapter 3). This algorithm solves the λ_e-QPM parametrically for all λ_e in the interval $[0, +\infty)$.
Starting from one point on the Efficient Frontier, the algorithm computes a sequence of so called *corner portfolios* $\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}$. These corner portfolios define the complete Efficient Frontier as all other points on the Pareto front are convex combinations of the two adjacent corner portfolios:
If $\boldsymbol{x_i}$ and $\boldsymbol{x_{i+1}}$ are adjacent corner portfolios with expected returns E_i and E_{i+1}, $E_i \leq E_{i+1}$, then for every $E_{i,\lambda}$ with $E_{i,\lambda} = \lambda E_i + (1-\lambda)E_{i+1}$, $\lambda \in [0,1]$ the optimal portfolio $\boldsymbol{x_\lambda}$ is calculated as $\boldsymbol{x_\lambda} = \lambda \boldsymbol{x_i} + (1-\lambda)\boldsymbol{x_{i+1}}$.
Depending on the capabilities of the algorithm that is used to solve the mean-variance optimization problem, the linear constraints may have to be adapted to the required formulation. The "classical" parametric quadratic programming algorithm from Markowitz [Mar87], the *Critical Line Algorithm*, supports only equations and nonnegativity constraints, i.e. inequalities which ensure that variables remain positive. Therefore, general inequalities of the type $\boldsymbol{Ax} \leq \boldsymbol{b}$ have to be transformed into equations by using slack variables (see, e.g., Markowitz [Mar87] and especially Rudolf [Rud94]). This approach is problematic as it increases the problem size due to the additional variables.
In Chapter 3 we present an optimized version of a parametric quadratic programming algorithm that accepts both equations and inequalities. Thus, no modifications to the problem structure are necessary[4].

An important simplification common to all portfolio selection models mentioned above is that the elements of $\boldsymbol{x}$ are assumed to be real numbers. Considering that shares can usually not be bought and sold in fractions, this may have the effect that the solution of the SMVM (and therefore also solutions of the ϵ-QPM and the λ_e-QPM) may not be applicable to the actual optimization problem of the investor. The divergence can be quite significant if the available budget is small. Given a larger budget, however, the difference between the solution of problem SMVM and the solution of the actual optimization problem – where the number of traded assets has to be an integer – is negligible[5].
The computational difficulties that result if integer constraints are included in the optimization are briefly discussed in Section 2.5.2 together with problems caused by other types of constraints that can not be integrated into the standard model.

[4] Naturally, all variables have to be shifted to the left hand side of the constraints, and "larger than" inequalities have to be multiplied by -1.

[5] Another justification for using real valued variables is that in several interviews, portfolio managers for mutual funds did confirm that they only work with fractions of the available budget due to the fact that their budget is subject to daily changes.

2.3 Measures of Risk

One common interpretation of the variance in the standard model is that it quantifies portfolio risk. Therefore, mean-variance optimization is often referred to as risk-return optimization. This point of view is controversial as the meaning of the term *risk* in everyday perception clashes with the mathematical definition of variance. One main problem of variance as a measure of risk is that both positive and negative deviations of the actual return from the expected portfolio return are equally taken into account when the variance is calculated. Only very few investors will, however, consider it a problem if the portfolio return is larger than the return that was expected before. A risk measure that only measures the downside deviations while leaving out the upside potential may be more compatible with what would usually be expected from a measure of risk.
As a consequence of this problem and also due to the rising importance of risk management in financial institutions, which are also mainly concerned with negative deviations of the return, several authors have examined the application of alternative risk measures in portfolio selection with the intention to better capture the notion of risk.
Two approaches that have played a prominent role as risk measures in the last few years are Value at Risk (VaR) and Conditional Value at Risk (CVaR).
Value at Risk is a concept that describes risk as the loss of a portfolio of assets which is not surpassed given a confidence level α. The VaR is therefore the difference between the expected return of the portfolio and the $(1-\alpha)$-quantile of its return distribution. Due to certain drawbacks of the VaR-approach like, e.g., the lack of sub-additivity (see Artzner et al.[ADEH99]), Conditional Value at Risk is often suggested as a suitable replacement. Conditional Value at Risk (CVaR), also called Expected Tail Loss (ETL), is the expected loss under the condition that the portfolio return is below the same α-quantile that marks the threshold of the VaR.
There are other measures of risk that try to capture the asymmetric meaning of risk, with the semivariance measure suggested by Markowitz [Mar59] being the most prominent. Grootveld and Hallerbach [GH99] analyze different downside-risk measures and compare the results of their application to those of the standard mean-variance framework.

Konno and Yamazaki [KY91] proposed to replace portfolio variance with the so-called Mean Absolute Deviation (MAD) which is defined as follows:

$$\omega_p = E(\mid \boldsymbol{x}'\boldsymbol{r} - \boldsymbol{\mu}'\boldsymbol{x} \mid)$$

with $\boldsymbol{r}$ as the vector of random variables representing the returns of all assets, $\boldsymbol{x}$ as vector of portfolio weights and $\boldsymbol{\mu}$ as vector of expected returns. Their main arguments for this modification were:

1. With MAD, no covariance matrix is necessary. Therefore the number of parameters to be estimated before the optimization is significantly lower.

2. Konno and Yamazaki claim that quadratic mean-variance optimization with large dense covariance matrices is computationally not feasible. In the MAD model, it is only necessary to solve a linear optimization problem.

3. They also argue that the solution of their optimization model results in fewer assets being included in the portfolio whereas for the mean-variance case the number of assets with a weight larger than 0 may be large.

The second reason does not align with our experience (cf. Chapter 3), and we also did not witness the effect that the mean-variance model results in a large number of small weighted assets. Simaan [Sim97] compared both models with respect to the consequences of estimation errors in the parameters. He concluded that the resulting error is less severe in the mean-variance model.
In this thesis, neither VaR, nor CVaR, nor MAD will be considered any further. Instead, we will focus on the variance of the portfolio return. For additional information on VaR and CVaR in the context of portfolio selection, the reader is referred to e.g. Uryasev [Ury00], Krokhmal et al. [KPU02], Maringer [Mar05], Gaivoronsky and Pflug [GP05], or Alexander and Baptista [AB04].
Besides the initial paper from Konno and Yamazaki [KY91], the MAD model is discussed and extended in Konno and Wijayanayake [KW02] and different solution approaches are treated in Konno and Yamamoto [KY05]. Mansini et al. [MOS03] give an overview of the different LP-solvable portfolio optimization problems, among them the MAD and the CVaR model. They also provide a computational comparison of the different models on real life data.

2.4 Benchmark Problems

Many authors test their approaches on the publicly available benchmarks provided in the OR-library [Bea06]. For mean-variance portfolio selection, 5 data sets are available, which we will use as well, namely

P1 The smallest problem, the Hang Seng benchmark consisting of 31 assets.

P2 The benchmark data set based on the DAX 100 containing 85 assets.

P3 The benchmark based on the FTSE 100 with 89 assets.

P4 The S&P benchmark with 98 assets.

P5 The largest problem in the OR-library, the Nikkei 225 benchmark with 225 assets.

The data sets consist of values for the expected return and the standard deviation of each asset and of the correlation matrix. They were initially used by Chang et al. [CMBS00]. Since we also required larger data sets in order to examine which approaches scale well, we additionally tested it on larger problem instances:

P6 A benchmark with 500 assets

P7 A benchmark with 1000 assets

P8 A benchmark with 2000 assets

The data sets were generously provided by the authors of Hirschberger et al. [HQS07]. They were generated according to a method described in that paper.

2.5 Constraint Types

In this section, we will discuss (i) constraints that are relevant for German mutual funds and (ii) constraints that are mentioned in publications in this field of research.
With the Standard Mean-Variance Model (SMVM) in mind, constraints for mean-variance portfolio selection can be divided into two groups: constraints that are compatible with the model and those that are not.
A constraint is compatible with the SMVM if it can be expressed as a linear term of the optimization variables which has to be either larger than, smaller than, or equal to a given constant:

$$a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n \oplus b$$

with $\oplus \in \{\leq, \geq, =\}$.
Any feasible region defined by a single linear constraint or the intersection of an arbitrary number of linear constraints is convex.

Definition. Convex Set:
A set M is *convex* if, for any two points $\boldsymbol{x}, \boldsymbol{y} \in M$, all points $\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y},\ \lambda \in [0,1]$ are elements of M as well.

Definition. Convex Function:
A function defined on a convex set M is convex if for any two point $\boldsymbol{x}, \boldsymbol{y} \in M$ the following is true for all $\lambda \in [0,1]$:

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda)f(\boldsymbol{y})$$

Convexity of the search space is, together with a convex objective function, *the* essential property that makes an optimization problem easier to solve. If both requirements are fulfilled, then each local minimum of the objective function is also a global minimum. It is therefore sufficient to find a local minimum, which is usually not too difficult. The obvious approach is to find a solution that fulfills the Karush-Kuhn-Tucker conditions.
There are, however, several types of constraints that are convex but cannot be expressed as linear constraints. One such constraint would be an upper bound on the variance of the portfolio. All parametric and most nonparametric quadratic programming solvers are, as of today, unable to cope with such constraints, although in the last few years, quadratic constraints have been integrated in a few commercial packages (c.f. [NEO06]).
All three models presented in Section 2.2 use the variance of the portfolio as a part of or as the sole optimization criterion. We think that as a consequence of this, an additional constraint that restricts variance does not make sense. Other types of constraints that are convex but not linear are – to our knowledge – not broadly used in the context of portfolio selection.
The constraints that are discussed in the remainder of this section are either linear or nonconvex.

2.5.1 Linear Constraints

Most of the restrictions that are relevant for portfolio selection can be expressed as linear constraints. The following list provides the most common types of linear constraints:

1. The prohibition of short sales[6]:
   For many mutual funds, short selling is not allowed at all. In the context of a single-period mean-variance framework, such a constraint is usually modelled as $\boldsymbol{x} \geq 0$.

2. Lower bounds on individual assets:
   Constraints of this type are modelled as $x_i \geq l_i$ and are therefore just a generic variant of the short sales prohibition.

3. Upper bounds on single asset ($x_i \leq u_i$), on groups of single assets ($x_i \leq u_i \; \forall i \in \Upsilon$), or on all assets ($\boldsymbol{x} \leq \boldsymbol{u}$).
   If there are upper bounds on all assets, such constraints have the effect to prevent the uneven distribution of the budget among only very few assets. Constraints of this type are often defined by laws which intend to limit the leeway of those responsible for the investment policy of financial instruments. In a German mutual fund, for example, the share of the budget that is invested into any single asset is not allowed to surpass 10% (see [Inv05]). Upper bounds on single assets that are lower than those on all the other assets may occur if the investor has a certain aversion against them. This may go as far as to set the upper and lower bound of an asset to the identical value. In this case, the fraction invested in this asset is already determined and does not have to be calculated. The dimension of the optimization problem can (and should) therefore be reduced by 1.

4. Sector constraints:
   Constraints of this type limit the fraction of the budget that may be invested in certain sectors, e.g. in assets of automobile producers, in the energy sector, or in shares from software producers. A sector constraint can be modelled as follows:

$$\sum_{i \in \Xi} a_i x_i \leq b$$

   where Ξ is the set of those assets that belong to the restricted sector. a_i is usually 1, but constraints are conceivable where for some $i \in \Xi$, a_i is increased or decreased. One plausible situation where this might be the case is if a corporation is active in several sectors, only the "parts" of the corporation active in a particular sector are relevant for the sector constraint.

   Lower bounds on sectors are sometimes defined as well ($\sum_{i \in \Xi} a_i x_i \geq b$). Constraints of this type are usually used if a mutual fund is dedicated to a particular

[6] Short selling allows an investor to profit from falling prices by "borrowing" an asset and selling it with the expectation that he can buy the same asset back later for a reduced prize.

sector like, e.g., the oil industry, and the prospectus defines a guarantee that a given share of the budget is invested in this sector at all times.

5. Regional constraints:
   Constraints of this type set an upper or lower bound to the fraction of the budget that may be invested in assets of a given region, for example
   At least 70% of the budget has to be invested in European shares. They are modelled similarly to sector constraints.

   Both sector constraints and regional constraints are often specified in the prospectus of an investment fund, as they define its overall investment policy.

The consequences of any of these constraints depend strongly on the actual data. What can be stated irrespective of the type of constraint is that if the basic tenets of mean-variance optimization are accepted and the given problem data (the vector of expected returns and the covariance matrix) is assumed to be correct, every optimization problem with any type of additional constraint can at best have a solution of the same quality as the equivalent problem that does not include the constraint. If the constraint is binding, the result will often be worse. As a consequence, it does not make sense for an investor to introduce too many restrictions.
The main argument for the introduction of constraints is that they keep the solution of the mean-variance model in an area of the search space that is economically reasonable[7].

2.5.2 Nonconvex Constraints

In the following we will describe several constraints that can not be expressed in a way which would permit their integration into the models presented in Section 2.2, i.e. they cannot be expressed as a set of linear equations and/or linear inequalities. The core problem with all of them is that they cause the search space to become nonconvex.

Buy-In Thresholds

Buy-in thresholds prevent assets from being included in a portfolio with small weights only. They determine that asset weights are either above a lower bound l_b, or the asset is not part of the portfolio at all. The main reason for such a constraint might be that some costs are – at least partially – determined by the number of different shares that are held (e.g. information costs, fixed transaction costs). Shares with very small weights would therefore just increase these costs without having any real impact regarding portfolio variance or expected return.
Jobst et al. [JHLM01] have shown that a portfolio selection problem with buy-in thresholds can be formulated as a *mixed-integer problem* (MIP)[8] by adding a binary variable

[7] If the problem is completely unconstrained, i.e. if not even the amount an asset can be sold short is limited, it is possible to achieve infinite expected return. In that case, an infinite amount of stock has to be bought and sold short, however. Such a result is not compatible with economic reality.

[8] Mixed-integer problems, often also called mixed-integer programs, are optimization problems in which some of the decision variables have to be integral while others are allowed to assume real values.

for each investment alternative. With N assets and binary variables $\rho_i,\ i = 1 \dots N$, the MIP looks as follows:

SMVM with buy-in threshold l_b

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{2.7a}$$

$$\max E(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\mu} \tag{2.7b}$$

subject to

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.7c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{2.7d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.7e}$$

$$l_b \rho_i \leq x_i \leq u \rho_i \tag{2.7f}$$

$$\rho_i \in \{0, 1\} \quad i = 1 \dots N \tag{2.7g}$$

u denotes the upper bound for all assets. It can be set to 1 if no stricter upper bound is given.
Unfortunately, neither "standard" QP solvers nor their parametric variants can handle binary variables, as such variables are inherently nonconvex. But even if it is possible to model the problem without binary variables, the search space of this problem is nonconvex due to the added constraint.

Proof. Assume the portfolios $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible, and that variable $x_i = 0$, and variable $y_i = l_b$. Obviously any convex combination of $\boldsymbol{x}$, $\boldsymbol{y}$, apart from $\boldsymbol{x}$ and $\boldsymbol{y}$ themselves, is not feasible. □

Cardinality Constraints

Investors and fund managers often wish to control the number of assets in the mutual fund they own/manage:
The fund manager might – due to monitoring reasons, or in order to reduce transaction costs – set an upper limit on the number of securities in a portfolio (*maximum cardinality constraint*).
A wealthy private investor, however, might prefer a well diversified portfolio, and therefore sets a number of assets which his portfolio must contain at least (*minimum cardinality constraint*). Horniman et al. [JHLM01] pointed out that a minimum cardinality constraint is intrinsically linked with buy-in thresholds: on the one hand, a high threshold limits the number of assets the portfolio can contain, and on the other hand, if there is no threshold at all, any x_i can be set to a very small value instead of being zero, meaning that the minimum cardinality constraint is rendered completely ineffective.
The nonconvexity of the search space prevents the application of the classical methods (e.g. parametric quadratic programming) on problem instances with maximum cardinality constraints as well.

Proof. If a maximum cardinality of K is assumed, and it is further assumed that the portfolios $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible, and that variable $x_i > 0$ for $i = 1, \ldots, K$ and that variable $y_i > 0$ for $i = K+1, \ldots, 2K$, then any convex combination of $\boldsymbol{x}$, $\boldsymbol{y}$, apart from $\boldsymbol{x}$ and $\boldsymbol{y}$ themselves, is not feasible. □

The nonconvexity of the search space in a problem with constrained minimum cardinality stems from the buy-in thresholds that are necessary in this case.

By using integer variables, the Standard Mean-Variance Model (SMVM) can be modified so that its solution complies with a maximum cardinality constraint. If we assume that the number of different assets to be held is limited to K, the model looks as follows:

SMVM with maximum cardinality constraint

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{2.8a}$$

$$\max E(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\mu} \tag{2.8b}$$

subject to

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.8c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{2.8d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.8e}$$

$$x_i \leq u_i \rho_i \qquad \rho_i \in \{0, 1\} \qquad i = 1, \ldots, N \tag{2.8f}$$

$$\sum_{i=1}^{n} \rho_i \ \leq \ K \tag{2.8g}$$

Ineq. 2.8f requires an upper bound u_i for every variable. If none is given, it can be set to 1 as long as short sales are not permitted. Otherwise, the upper bound has to be adapted accordingly – although it is questionable if it makes much sense to use a cardinality constraint when short sales are allowed.
Irrespective of how the maximum cardinality constraint is modelled, we would like to point out that the presence of such a constraint can make the problem NP-complete, but only if the maximum cardinality is a fraction of the total number of assets (Bienstock [Bie96]). If the maximum cardinality is a small constant, the number of different combinations of shares larger than 0 is polynomial.

The 5-10-40-Constraint

The German investment law [Inv05] states in §60(1) that, roughly translated, securities of the same issuer are allowed to amount to up to 5% of the net asset value of the mutual fund. They are allowed to amount to 10%, however, if the total of all of these assets is less than 40% of the net asset value. This constraint is especially interesting as it is the only one based on German law that can't be incorporated into the SMVM (see Section 2.2) in the form of linear constraints.
Again, the difficulty is the result of the nonconvexity of the search space / feasible region.

Proof. Assume that the portfolios $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible, and that $x_i = 0.1,\ i = 1, \ldots, 4$ and $x_j = 0.05,\ j = 5, \ldots, 16$. Assume further that $y_i = x_i$ for all i except $y_4 = 0.05$ and $x_5 = 0.1$. Any convex combination of $\boldsymbol{x}$ and $\boldsymbol{y}$, apart from $\boldsymbol{x}$ and $\boldsymbol{y}$ themselves, is not feasible: the sum of all assets with $x_i > 0.05$ is obviously larger than 40%. □

As a consequence, this problem can't be handled by either a QP-solver or the available algorithms for parametric quadratic programming.
Using the vector $\boldsymbol{\rho}$ consisting of binary variables $\rho_i,\ i = 1, \ldots, N$, the portfolio selection problem with 5-10-40-Constraint can be formulated as follows:

SMVM with 5-10-40-Constraint

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{2.9a}$$
$$\max E(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\mu} \tag{2.9b}$$
subject to
$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{2.9c}$$
$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{2.9d}$$
$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{2.9e}$$
$$\boldsymbol{\rho}^T \boldsymbol{x} \leq 0.4 \tag{2.9f}$$
$$\boldsymbol{x} - 0.05\boldsymbol{\rho} \leq 0.05\boldsymbol{e} \tag{2.9g}$$
$$\rho_i \in \{0, 1\}\ \forall i = 1, \ldots, N \tag{2.9h}$$
$$\boldsymbol{x} \geq \boldsymbol{0} \tag{2.9i}$$

Ineq. 2.9g enforces that if $\rho_i = 1$, x_i may go up to 10%, otherwise ($\rho_i = 0$) the upper bound of x_i is 5%. Ineq. 2.9f then ensures that the combined share of those variables whose weight may become larger than 5% is limited to 40%.

Integer Constraints

Integer constraints, sometimes also called *minimum transaction lots* or *roundlots*, are another type of "complex" constraint often mentioned in publications that treat non-convex extensions of the mean-variance portfolio selection model (see e.g. Mansini and Speranza [MS99], Jobst et al. [JHLM01]). The reason for integer constraints is that many investment alternatives can usually not be bought and sold in non-integer quantities[9]. If the basic SMVM is used, element i of the solution will only by pure chance be an integer multiple of the budget fraction γ_i that is the equivalent of the prize for one share of investment alternative i.

[9] Exceptions are e.g. mutual funds and index certificates.

For the whole solution to comply with an integer constraint, x_i has to be replaced by $\gamma_i k_i$ in the SMVM:

SMVM-Int

$$\min V(\boldsymbol{k}) = (\boldsymbol{\Gamma k})^T \boldsymbol{C \Gamma k} \tag{2.10a}$$

$$\max E(\boldsymbol{k}) = (\boldsymbol{\Gamma k})^T \boldsymbol{\mu} \tag{2.10b}$$

subject to

$$\boldsymbol{A_I}(\boldsymbol{\Gamma k}) \leq \boldsymbol{b_I} \tag{2.10c}$$

$$(\boldsymbol{\Gamma k})^T \boldsymbol{e} \leq 1 \tag{2.10d}$$

$$\boldsymbol{A_E}(\boldsymbol{\Gamma k}) = \boldsymbol{b_E} \tag{2.10e}$$

$$k_i \text{ integer} \quad \forall i \tag{2.10f}$$

where $\boldsymbol{k}$ is the vector of integer decision variables. To simplify the notation, we have introduced the $N \times N$-matrix $\boldsymbol{\Gamma}$, a diagonal matrix with the diagonal elements consisting of vector $\boldsymbol{\gamma}$. Ineq. 2.10d had to be modified in order to cope with leftover budget that is caused by the integer constraints. Another way to handle this would be to introduce an asset that represents cash and is divisible up to the smallest currency unit (e.g. cent).
Based on empirical results, Maringer [Mar02a] has concluded that integer constraints are primarily relevant for investors with a small budget, as the difference in results between an integer solution and the solution of the basic model becomes insignificant when the budget is increased. As our main intention is the treatment of constraints that are relevant for investors with large budgets, we will not consider integer constraints in the remainder of this thesis.
Instead, the focus will be put on the development of solutions for problems containing cardinality constraints, the 5-10-40-Constraint, and partially for problems with buy-in thresholds.
For further information about methods that allow the calculation of solutions for problems with integer constraints, the reader is referred to Mansini and Speranza [MS99], Jobst et al. [JHLM01], Lin and Wang [LW02], Maringer [Mar02a], Streichert et al. [SUZ03, SUZ04a, SUZ04b], and Li et al. [LSW06].

3 A Solver for Parametric Quadratic Programming

This chapter aims at providing an efficient and reliable algorithm that is able to calculate the complete Pareto front for the mean-variance portfolio selection problem with linear equations and inequalities. For further background information about the mean-variance problem, the reader is referred to Section 2.2.
There are several reasons why it may be necessary to calculate the complete Pareto front:

- Not much is known about the utility function or the risk aversion of an investor.
- The investor wants to take a look at all "interesting" portfolios and then, based on that, she will take her decision.
- There is a large group of investors with similar constraints but different risk-return preferences.

As mentioned in Chapter 2, algorithms for the calculation of all portfolios on the Pareto front belong to the category of parametric quadratic programming algorithms – or shorter: PQP algorithms – and are presented in several publications. The first of them was the so called *Critical Line Algorithm* mentioned in the seminal work by Markowitz [Mar56].
To our knowledge, all algorithms that are capable of calculating the whole Efficient Frontier assume that the search space is convex, usually by requiring that all constraints are linear in nature. (For a brief overview of the different types of constraints, see Section 2.5.)
As the overall goal of this thesis is the handling of nonconvex constraints in a mean-variance framework, a PQP algorithm is unable to solve problems with such constraints directly and does not help much at first glance. However, by selecting different convex subsets in the nonconvex search space, we are able to use a PQP algorithm to calculate a Pareto front for each subset. These Pareto fronts can then be merged into a solution for the problem with nonconvex restrictions. For a detailed description on how the convex subsets are chosen, the reader is referred to Chapter 5 and Chapter 6. Chapter 5 also describes how a PQP algorithm can be integrated into an evolutionary algorithm framework. For such an application, efficiency of the PQP algorithm is particularly crucial, as it has to be executed multiple times. A faster algorithm permits a larger number of convex subsets to be calculated, which in turn might lead to a better overall solution.

Only very few publications that describe algorithms for parametric quadratic programming give any hints on how to implement these algorithms in an efficient and numerically stable way for large portfolio sizes, which is our main focus in the remainder of this chapter. The chapter is based on an article by Stein, Branke, and Schmeck [SBS07].

The subsequent sections of this chapter are organized as follows:
Section 3.1 presents the algorithm framework that provides the basis of our implementation. In Section 3.2 we compare other existing approaches for parametric and non-parametric quadratic programming. Based on our framework from Section 3.1 several modifications intended to shorten algorithm runtime will be described in Section 3.3. The test results for these algorithm variants are presented in Section 3.4. In Section 3.5, two different matrix representations are introduced, and we show how both are incorporated into the implementation of the fastest algorithm variant. We also highlight one implementation detail that is crucial to achieve correct solutions. As there is no publicly available algorithm that realizes all the proposed improvements, in Section 3.6 we provide a step-by-step description of the fastest algorithm variant which should help the reader in implementing it efficiently. (Our own implementation will be used in Chapter 5 to calculate the solutions for the convex subsets.) The chapter concludes with a brief summary and hints at potential extensions of the PQP algorithm.

3.1 An Algorithm for Parametric Quadratic Programming

As a starting point, we have used a modified version of the active set algorithm for parametric quadratic programming (PQP) presented by Best [Bes96] and have adapted it for portfolio selection. In order to familiarize the reader with all the variables and the used methodology, we summarize the algorithm in a short way and refer the reader to Best [Bes96] for a more detailed description. The implemented algorithm solves the λ_e-QP model (see Section 2.2) for all values of the parameter λ_e in the interval $[0, +\infty)$:

λ_e-QPM

$$\min F(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} - \lambda_e \boldsymbol{x}^T \boldsymbol{\mu} \tag{3.1a}$$

subject to

$$\boldsymbol{A_I} \boldsymbol{x} \leq \boldsymbol{b_I} \tag{3.1b}$$

$$\boldsymbol{A_E} \boldsymbol{x} \leq \boldsymbol{b_E} \tag{3.1c}$$

with the element x_i of the n-vector $\boldsymbol{x}$ again denoting the fraction of the budget invested in asset i. The $n \times n$-matrix $\boldsymbol{C}$ is the covariance matrix, the n-vector $\boldsymbol{\mu}$ denotes the vector of expected returns of all assets. $\boldsymbol{A_I}$ and $\boldsymbol{A_E}$ are the coefficient matrices of inequalities and equalities; $\boldsymbol{b_I}$ and $\boldsymbol{b_E}$ denote the respective right hand sides. To simplify notation, the budget constraint ($\boldsymbol{x}^T \boldsymbol{e} = 1$) is considered as part of the equations and will not be mentioned separately in the remainder of this chapter.

To start the parametric programming routine, at least one portfolio on the Pareto front has to be known. Due to the fact that it is usually easier and computationally cheaper to solve an optimization problem with a linear objective function instead of a quadratic objective function, our PQP algorithm starts at the portfolio with the highest possible expected return, which is the solution of the following optimization problem:

$$\max\{\boldsymbol{\mu}^T\boldsymbol{x} \mid \boldsymbol{A_I x} \leq \boldsymbol{b_I},\ \boldsymbol{A_E x} = \boldsymbol{b_E}\} \tag{3.2}$$

If this solution is unique, as is nearly always the case for "normal" portfolio selection problems, the portfolio lies at the end of the Efficient Frontier that is associated with the highest λ_e.
Otherwise, there are infinitely many portfolios that achieve the highest possible expected return, and it is necessary to select from all these solutions to problem (3.2) the portfolio with the lowest variance:

$$\min\{\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} \mid \boldsymbol{\mu}^T\boldsymbol{x} = E_m, \boldsymbol{A_I x} \leq \boldsymbol{b_I},\ \boldsymbol{A_E x} = \boldsymbol{b_E}\} \tag{3.3}$$

Here, E_m denotes the maximum expected return calculated in (3.2). This is a quadratic programming problem that can be solved with any of the available standard codes or packages for this problem class (see e.g. the NEOS Optimization Software Guide [NEO06]).
If not all portfolio weights have an upper and lower bound, it is possible that the expected return is unbounded and therefore a solution to problem (3.2) does not exist. In this case it is necessary to compute the minimum variance portfolio:

$$\min\{\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} \mid \boldsymbol{A_I x} \leq \boldsymbol{b_I},\ \boldsymbol{A_E x} = \boldsymbol{b_E}\} \tag{3.4}$$

and start the parametric quadratic programming algorithm from the other end of the Efficient Frontier.
For the PQP algorithm to successfully calculate the global optimum for each λ_e, it is necessary to have a positive semidefinite covariance matrix $\boldsymbol{C}$. For portfolio selection problems this is always the case due to the fact that the variance $V = \boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}$ of any portfolio has to be larger than or equal to zero[1].
At the heart of the active set algorithm is the separation of the inequalities into active and inactive constraints at a given point $\hat{\boldsymbol{x}}$. A linear inequality $f(\hat{\boldsymbol{x}}) \leq b$ is active at $\hat{\boldsymbol{x}}$, if the equation $f(\hat{\boldsymbol{x}}) = b$ is fulfilled as well. Otherwise (i.e. if $f(\hat{\boldsymbol{x}}) < b$) the constraint is inactive. Equations can be interpreted as constraints that are always active. The set of all constraints (both inequalities and equations) active at a given point is called the *active set*. The Efficient Frontier consists of adjacent segments, each of which is characterized by an active set I_a^n that is constant. At the transit point – also called *corner portfolio* – to the next segment, the active set changes. The first constraint that switches status determines both the end of the segment and the set I_a^{n+1} of the adjacent segment.

[1] Difficulties might arise if the covariance matrix is positive semidefinite, but not positive definite, as even slight numerical errors can cause the matrix to become indefinite. To prevent this, it might be advisable to add a very small positive constant (i.e. 10^{-7}) to all diagonal elements.

One iteration of the algorithm works as follows:
If λ_e of the starting point and I_a^n for the current segment are given, and $\boldsymbol{A_a}$ is the matrix of active constraints, the end of the segment is computed by solving the system of linear equations that results from the Karush-Kuhn-Tucker conditions

$$\begin{pmatrix} \boldsymbol{C} & \boldsymbol{A_a^T} \\ \boldsymbol{A_a} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b_a} \end{pmatrix} + \lambda_e \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{0} \end{pmatrix} \tag{3.5}$$

and determining the first Lagrange multiplier λ_i of any inequality that becomes 0. (The Lagrange multipliers of the equations are allowed to be negative.) Additionally the inactive constraints have to be checked whether they become active. If the matrix $\boldsymbol{H} = \left(\begin{smallmatrix} \boldsymbol{C} & \boldsymbol{A_a^T} \\ \boldsymbol{A_a} & \boldsymbol{0} \end{smallmatrix}\right)$ is nonsingular, the parametric solution of (3.5) can be determined by solving the two sets of linear equations:

$$\boldsymbol{H} \begin{pmatrix} \boldsymbol{h_{x,q}} \\ \boldsymbol{h_{\lambda,q}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b_a} \end{pmatrix}$$

and

$$\boldsymbol{H} \begin{pmatrix} \boldsymbol{h_{x,p}} \\ \boldsymbol{h_{\lambda,p}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{0} \end{pmatrix}$$

for the vectors $\left(\begin{smallmatrix} \boldsymbol{h_{x,q}} \\ \boldsymbol{h_{\lambda,q}} \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} \boldsymbol{h_{x,p}} \\ \boldsymbol{h_{\lambda,p}} \end{smallmatrix}\right)$.
To determine the end of the current segment, the values of $\hat{\lambda}_e$ and $\tilde{\lambda}_e$ have to be obtained. $\hat{\lambda}_e$ indicates the end of the current segment if the end point is determined by any of the inactive constraints becoming active. $\hat{\lambda}_e$ is calculated as follows:

$$\hat{\lambda}_e = \max \left\{ \frac{b_i - \boldsymbol{a_i}^T \boldsymbol{h_{x,q}}}{\boldsymbol{a_i}^T \boldsymbol{h_{x,p}}} \; \forall \; i = 1 \ldots m \text{ with } i \notin I_a \text{ and } \boldsymbol{a_i}^T \boldsymbol{h_{x,p}} < 0 \right\} \tag{3.6}$$

$\tilde{\lambda}_e$ indicates the end of the segment if the end point is determined by some restriction changing status from active to inactive:

$$\tilde{\lambda}_e = \max \left\{ \frac{h_{\lambda,q,i}}{h_{\lambda,p,i}} \; \forall \; i = 1 \ldots m \text{ with } i \in I_a \text{ and } h_{\lambda,p,i} > 0 \right\} \tag{3.7}$$

λ_e for the next corner portfolio is therefore calculated as $\lambda_e = \max\{\hat{\lambda}_e, \tilde{\lambda}_e\}$. If λ_e is determined by (3.6) and the responsible index is l, $I_a^{n+1} = I_a^n \cup \{l\}$. Similarly, if λ_e is determined by (3.7) and the responsible index is o, $I_a^{n+1} = I_a^n \setminus \{o\}$. Both Best [Bes96] and Perold [Per84] present a similar solution for what to do if row l of matrix $\boldsymbol{A}$ which has to be added to $\boldsymbol{A_a}$ is linearly dependent on $\boldsymbol{A_a}$. They then calculate the constraint that changes from active to inactive and remove the constraint from the active set I_a^{n+1}. In both algorithms, however, there is the prerequisite that the current matrix $\boldsymbol{A_a^n}$ has full row rank.
If the algorithm starts at the point with the highest expected return, it happens quite often that the first A_a^0 constructed from this point does not comply with the above requirement.

Example:

$$\max E(\boldsymbol{x}) = \lambda_e \, \boldsymbol{\mu}^T \boldsymbol{x} \tag{3.8a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{3.8b}$$

$$\boldsymbol{x} \leq \mathbf{0.1} \tag{3.8c}$$

$$\boldsymbol{x} \geq \mathbf{0} \tag{3.8d}$$

The solution for this problem will have the ten shares with the highest expected return at their upper bound 0.1 and all other shares at their lower bound 0. Therefore, for each share either its upper bound or its lower bound constraint is active. The coefficient vector of budget constraint (3.8b) can then be expressed as a linear combination of the coefficient vectors of the upper and lower bounds which means that A_a does not have full row rank.
The solution we propose, which is even applicable if the covariance matrix is only positive semidefinite and not positive definite, works as follows:
If the matrix H is singular, the following optimization problem is solved to determine λ_e, $\boldsymbol{\lambda}$, and $\boldsymbol{x}$:

MIN-λ_e

$$\min \ \lambda_e^{n+1} \tag{3.9a}$$

subject to

$$\boldsymbol{C}\boldsymbol{x} + \boldsymbol{A}_{\boldsymbol{a}}^{\boldsymbol{T}}\boldsymbol{\lambda} = \ \lambda_e^{n+1}\boldsymbol{\mu} \tag{3.9b}$$

$$\boldsymbol{A}\boldsymbol{x} \leq \ \boldsymbol{b} \tag{3.9c}$$

$$\boldsymbol{\lambda} \geq \ \mathbf{0} \tag{3.9d}$$

The elements of the calculated $\boldsymbol{x}$ determine the weights at the end of this segment, elements of $\boldsymbol{\lambda}$ that are 0 indicate which of the active constraints will be inactive in the next segment. If any of the inactive constraints become active – which is easily tested by inserting the calculated $\boldsymbol{x}$ into (3.9c) – it is inserted into I_a^{n+1}.
By construction, this procedure guarantees Karush-Kuhn-Tucker optimality at the end of the current segment. (It is trivial to show that solutions for values of λ_e in the interval $(\lambda_e^{n+1}, \lambda_e^n)$ which are calculated as $\lambda_e = \xi\lambda_e^{n+1} + (1-\xi)\lambda_e^n$, $\xi \in (0,1)$ are optimal as well.)

The following short description of the algorithm concludes this section:

1. Initialization:
   a) Calculate and store the initial starting solution $\boldsymbol{x_0}$.
   b) Determine the associated active set I_a^0
   c) Set $k \longleftarrow 0$ and set $\lambda_e^k \longleftarrow \infty$.

2. If $\boldsymbol{H} = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{A_a^T} \\ \boldsymbol{A_a} & \boldsymbol{0} \end{pmatrix}$ is nonsingular
   a) Calculate the vectors $\begin{pmatrix} \boldsymbol{h_{x,q}} \\ \boldsymbol{h_{\lambda,q}} \end{pmatrix}$ and $\begin{pmatrix} \boldsymbol{h_{x,p}} \\ \boldsymbol{h_{\lambda,p}} \end{pmatrix}$
   b) Calculate the values $\hat{\lambda}_e$, $\tilde{\lambda}_e$ and $\lambda_e^{k+1} \longleftarrow \max\{\hat{\lambda}_e, \tilde{\lambda}_e\}$
   c) Set $\boldsymbol{x^{k+1}} \longleftarrow \boldsymbol{h_{x,q}} + \lambda_e^{k+1} \boldsymbol{h_{x,p}}$.
   d) Determine the next active set I_a^{k+1} and update matrix $\boldsymbol{H}$.

   Otherwise
   a) Calculate λ_e^{k+1}, $\boldsymbol{\lambda^{k+1}}$ and $\boldsymbol{x^{k+1}}$ by solving the linear optimization problem **MIN-**λ_e.
   b) Determine the next active set I_a^{n+1} and update matrix $\boldsymbol{H}$ accordingly.
3. If $\lambda_e^k > 0$, store the calculated solution $\boldsymbol{x^{k+1}}$, set $k \longleftarrow k+1$ and go to step 2, otherwise (i.e. if the end of the algorithm run is reached) calculate and store

$$\boldsymbol{x^{k+1}} \longleftarrow \frac{\lambda_e^k}{\lambda_e^k - \lambda_e^{k+1}} \boldsymbol{x^{k+1}} + \frac{\lambda_e^{k+1}}{\lambda_e^k - \lambda_e^{k+1}} \boldsymbol{x^k}$$

4. Terminate.

3.2 Related Work

The first and also – at least in the field of portfolio selection – best known algorithm for parametric quadratic programming was developed by Harry Markowitz [Mar56] and is called the Critical Line Algorithm (see e.g. the book from Markowitz [Mar87] for a detailed description). This algorithm in its basic form allows as constraints only equations and nonnegativity bounds for individual variables. To model additional inequalities, slack variables have to be added. This is a huge disadvantage in optimization problems with upper bounds for all assets, as the number of variables is at least doubled. (For an extensive description how the Markowitz algorithm has to be modified in order to cope with upper bounds, the reader is referred to Rudolf [Rud94].)
Perold [Per84] presented an algorithm version that is especially adapted to covariance matrices which are generated by a factor/index model (see e.g. Sharpe [Sha63] or Elton et al. [EGBG03]). This attribute of the covariance matrix makes it possible to express it in a way that permits the application of efficient factorization techniques for sparse matrix decomposition. The problem formulation considers upper and lower bounds on individual assets and also general equations. General inequalities require the introduction of slack variables. Additionally, the model from Perold allows the consideration of a so called turnover constraint, and it also permits the expected return to be the weighted sum of concave piecewise linear functions instead of just the weighted sum of linear functions. To speed up the algorithm further, Perold's procedure substitutes active upper and lower bounds into the system of equations that has to be solved (see Section 3.3).
The algorithm presented by Gould [Gou91] is actually a general purpose large-scale quadratic programming algorithm which is based on an active-set method. But Gould

also describes how to use his algorithm parametrically, although it is not specifically adapted to portfolio selection. It also uses the substitution of upper and lower bounds, and like the procedure from Perold, it is actually intended for sparse matrix representation.
Jacobs et al. [JLM05] integrated the factor technique used by Perold into the algorithm from Markowitz [Mar87]. The focus, however, was put on how to compute the Pareto front for portfolio selection problems with realistic short positions. The algorithm suffers from the same drawback as the Critical Line Algorithm from Markowitz, since upper bounds for assets require additional slack variables.
Best and Kale [BK00] also substituted active upper and lower bounds into the system of equations that has to be solved, but additionally, to further speed up the algorithm, they separated the resulting system of equations into two parts that are solved sequentially. Subsection 3.3.3 describes this in more detail.
Assumptions about the structure of the covariance matrix (besides the requirement of positive semidefiniteness) are not necessary for the algorithm variants we present in Section 3.3. One of their main advantages is the ability to work with dense covariance matrices without increasing computation time. (Sparsity does not play any role in the algorithm variants presented in the following section.)

"Normal" nonparametric primal active-set algorithms for quadratic programming and algorithms for parametric quadratic programming with a linear and a quadratic objective function do not differ too much. (Chapter 16 in the book from Nocedal [NW99] is a good introduction to active-set quadratic programming algorithms. The amount of published material on quadratic programming in general is immense, but the bibliography of Gould and Toint [GT01] contains an extensive listing of publications up to the year 2001.)
The main distinction is the requirement of the PQP algorithm to calculate all points on the Pareto front, while a nonparametric QP algorithm only needs to calculate the optimum for the single quadratic objective function. It is therefore more flexible in choosing the way how to proceed during an algorithm run. This advantage can be reflected in a reduced number of iterations, and therefore in faster algorithm convergence. In parametric quadratic programming, there is no shortcut: All corner portfolios have to be determined in order to be able to calculate all points on the Pareto front.
The operations performed on the matrix $\boldsymbol{H}$ and its decomposition, which are a consequence of either the activation of an inactive constraint or the deactivation of an active constraint, are essentially the same for both types of algorithms.
Where the algorithms mainly differ is in the computation of the initial solution and in the choice of pivoting operations. The nonparametric active-set algorithm can choose any active constraint to be dropped from the active set as long as the Lagrange multiplier has the correct sign, while the parametric algorithm needs to remove the first constraint whose multiplier would drop below 0 in order to maintain optimality.

3.3 Implementation Variants

The implementation variants of the algorithm framework we present in this section are aimed at calculating the Efficient Frontier as fast as possible without losing numerical stability. There are essentially two main options that offer the possibility to improve run time efficiency significantly:

1. The calculation of the initial decomposition of matrix $\boldsymbol{H}$.
2. The adaptation of the decomposition to the changes in the active set I_a.

It is only feasible to calculate the inverse of matrix $\boldsymbol{H}$ from scratch in every iteration if the problem size is very small. If we have n possible investment alternatives and m active constraints this would require $O((m+n)^3)$ floating point operations (flops) each time. For large problem sizes, it would be prohibitively slow in comparison to the use of a matrix decomposition and its consecutive adaption to the new active set. The decomposition of a matrix is an $O((n+m)^3)$ operation as well, at least if there is no specific knowledge about the structure of the matrix. It is sufficient, however, to calculate the decomposition once at the beginning of the algorithm run and to adapt it in each iteration to the new I_a. This adaptation – also called *updating* (if a constraint is added to the active set) or *downdating* (if a constraint is removed from the active set) – only requires $O((n+m)^2)$ flops.
The choice of the matrix decomposition which is used to solve System (3.5) is an important design decision. The most common decompositions which are applicable are the LU decomposition and the QR decomposition. The LU decomposition requires only about half the number of floating point operations for the initial decomposition. Its main disadvantage, however, is its numerical instability and that the up- or downdating is – if pivoting is used to improve numerical stability – complicated and requires more bookkeeping than the simpler updating procedures for the QR decomposition. There are efficient implementations for LU decompositions that try to maintain sparsity during up- and downdating procedures. This advantage is of great importance if the covariance matrix and the constraint matrices are sparse. As the following algorithm variants are intended to work well for general covariance matrices, however, this advantage does not pay off. We have therefore decided to use the QR decomposition.
But the techniques we present in the remainder of this section, which help to speed up the active set algorithm, can be applied to both decompositions.

3.3.1 Column Rearrangement

To improve algorithm efficiency it is essential to separate the inequality constraints into *upper* and *lower bounds* for single shares on one side and into *general inequalities* on the other side. It is our observation that in typical portfolio selection problems with a large number of investment alternatives, most of the inequalities are upper and lower bounds. The category of general inequalities often consists of sector or region constraints (e.g. constraints that limit the amount of money which is invested in European shares,

in energy sector shares, ...). If there are several hundred or even several thousand possible investment alternatives, the number of conceivable sector or region constraints will usually be small in comparison to the number of upper and lower bounds on single shares. As a consequence, for large problems, the set of constraints active at a given point on the Efficient Frontier is generally made up of mostly upper and lower bounds. This offers the possibility to significantly reduce the amount of work that is necessary to calculate the initial QR decomposition. As a first step the columns of the matrix $\boldsymbol{H}$ have to be rearranged:

$$\hat{\boldsymbol{H}} = \begin{pmatrix} \boldsymbol{A}_{\boldsymbol{b,a}}^{\boldsymbol{T}} & \boldsymbol{A}_{\boldsymbol{g,a}}^{\boldsymbol{T}} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_{\boldsymbol{g,a}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_{\boldsymbol{b,a}} \end{pmatrix}$$

$\boldsymbol{A}_{\boldsymbol{b,a}}$ denotes the coefficient matrix of the active bounds, $\boldsymbol{A}_{\boldsymbol{g,a}}$ is the coefficient matrix of the other (i.e. general) active constraints.
System (3.5) is changed accordingly:

$$\begin{pmatrix} \boldsymbol{A}_{\boldsymbol{b,a}}^{\boldsymbol{T}} & \boldsymbol{A}_{\boldsymbol{g,a}}^{\boldsymbol{T}} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_{\boldsymbol{g,a}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_{\boldsymbol{b,a}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda_b} \\ \boldsymbol{\lambda_g} \\ \boldsymbol{x} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b}_{\boldsymbol{g,a}} \\ \boldsymbol{b}_{\boldsymbol{b,a}} \end{pmatrix} + \lambda_e \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{pmatrix} \tag{3.10}$$

$\boldsymbol{\lambda_b}$ denotes the vector of Lagrange multipliers for the active bounds, $\boldsymbol{\lambda_g}$ represents the multipliers for the general constraints, $\boldsymbol{b}_{\boldsymbol{b,a}}$ and $\boldsymbol{b}_{\boldsymbol{b,g}}$ are the respective right hand sides of the constraints.
In order to minimize the number of flops required to transform $\hat{\boldsymbol{H}}$ into an upper triangular matrix, the active upper and lower bounds in $\boldsymbol{A}_{\boldsymbol{b,a}}$ have to be sorted beginning with the active bound with the lowest number and ending with the one with the hightest number.

Example:

In a problem with 5 shares and an initial efficient point where share 1 and 5 are at their upper bound, and share 2 is at its lower bound, $\boldsymbol{A}_{\boldsymbol{b,a}}$ looks as follows:

$$\boldsymbol{A}_{\boldsymbol{b,a}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(If all inequalities are formulated as less-or-equal constraints, all coefficients on the left hand side of a lower bound constraint flip their sign. The only nonzero coefficient in the second row is therefore -1.)
As matrix $\boldsymbol{H}$ is – due to the sorting of the constraints in $\boldsymbol{A}_{\boldsymbol{b,a}}$ – already well on its way to being triangular, it is easy to finish this process by using a sequence of Givens' rotations. The number of necessary Givens' (or plane) rotations to complete the QR decomposition depends on the number of active bounds ν and on the number of other (general) constraints ξ that are active.

If the matrix $\boldsymbol{H}$ is nonsingular, the number of active constraints $\nu+\xi$ has to be smaller than or equal to n. (If $\xi+\nu>n$, the matrix $\binom{\boldsymbol{A_{g,a}}}{\boldsymbol{A_{b,a}}}$ has more rows than columns and therefore cannot have full row rank. As a consequence, $\boldsymbol{H}$ is singular.)
The number of plane rotations required to bring $\boldsymbol{A_{b,a}}$ and $\boldsymbol{A_{b,a}^T}$ into the desired shape is $\nu(n-\nu)$. This is also the maximum number of rotations that are required to transform either $\boldsymbol{A_{b,a}}$ or $\boldsymbol{A_{b,a}^T}$, as the following instance of $\boldsymbol{A_{b,a}}$ shows:

$$\overbrace{\begin{matrix}x&0&0&0\\0&x&0&0\\0&0&x&0\\0&0&0&x\end{matrix}}^{\nu}\overbrace{\begin{matrix}x&x&x\\x&x&x\\x&x&x\\x&x&x\end{matrix}}^{n-\nu}\longrightarrow\overbrace{\begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\end{matrix}}^{n-\nu}\overbrace{\begin{matrix}x&0&0&0\\0&x&0&0\\0&0&x&0\\0&0&0&x\end{matrix}}^{\nu}$$

Fortunately, if matrix $\boldsymbol{A_{b,a}}$ has the shape that requires the most rotations, which occurs when bounds of the first ν variables are active, matrix $\boldsymbol{A_{b,a}^T}$ is already upper triangular and vice versa.

If the worst case for the matrix $\boldsymbol{A_{g,a}}$ and its transposed $\boldsymbol{A_{g,a}^T}$ is assumed, which is when the first and the last column of $\boldsymbol{A_{g,a}}$ have only nonzero elements, the number of Givens rotations necessary to transform both of them to the required shape is $\xi((n-\nu-1)+(n-\nu-\xi))$.

$$\overbrace{\begin{matrix}x&x&x&x&x&x&x&x\\x&x&x&x&x&x&x&x\\x&x&x&x&x&x&x&x\end{matrix}}^{n}\longrightarrow\overbrace{\begin{matrix}0&x&x&x\\0&0&x&x\\0&\underbrace{0}_{\xi}&0&x\end{matrix}}^{n-\nu}\overbrace{\begin{matrix}x&x&x&x\\x&x&x&x\\x&x&x&x\end{matrix}}^{\nu}$$

The last remaining part of $\boldsymbol{H}$ to be transformed is the covariance matrix $\boldsymbol{C}$ which requires $\frac{1}{2}(n-\nu-\xi-1)(n-\nu-\xi)$ plane rotations:

$$\underbrace{\begin{matrix}x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\end{matrix}}_{n}\longrightarrow\underbrace{\begin{matrix}x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\x&x&x&x&x&x&x\\0&x&x&x&x&x&x\\0&0&x&x&x&x&x\end{matrix}}_{n-\xi-\nu\quad\xi\quad\nu}$$

The upper bound s for the number of necessary plane rotations is therefore:

$$s=\nu(n-\nu)+\xi((n-\nu-1)+(n-\nu-\xi))+\frac{1}{2}(n-\nu-\xi-1)(n-\nu-\xi)\qquad(3.11)$$

With our assumption that most of the shares are either at their upper or lower bound, $n-\nu$ can be replaced by a (small) constant c, and equation (3.11) becomes:

$$s=\nu c+\xi((c-1)+(c-\xi))+\frac{1}{2}(c-\xi-1)(c-\xi)\qquad(3.12)$$

This indicates clearly that – if the assumption $\nu \approx n$ is valid – the number s of Givens rotations necessary to make $\boldsymbol{H}$ upper triangular is of order $O(n)$.
The technique we presented is also applicable to other constraints if the nonzero elements can be arranged in a proper way. The amount of saved computation time depends on the number of such constraints and on the number of nonzero elements below the main diagonal.

3.3.2 Substitution of Variables with Active Bounds

Another modification to greatly improve the runtime of the active set algorithm also uses the difference between simple bounds on individual assets and all other constraints. The basic idea – presented e.g. by Perold [Per84] or Gould [Gou91]) – is quite simple:
If either upper or lower bound on an asset is active in a given segment, then the variable associated with this constraint is fixed at this bound. Therefore the calculation of the variable in this segment is unnecessary, and the value of the bound can be substituted into the system of linear equations.
System 3.10 is then reduced to:

$$\begin{pmatrix} \boldsymbol{A}_{\boldsymbol{b,a}}^{\boldsymbol{T}} & \boldsymbol{A}_{\boldsymbol{g,a}}^{\boldsymbol{T}} & \boldsymbol{C'} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}'_{\boldsymbol{g,a}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda_b} \\ \boldsymbol{\lambda_g} \\ \boldsymbol{x'} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} + \boldsymbol{\alpha} \\ \boldsymbol{b_{g,a}} + \boldsymbol{\beta} \end{pmatrix} + \lambda_e \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{0} \end{pmatrix} \tag{3.13}$$

$\boldsymbol{C'}$ and $\boldsymbol{A'_{g,a}}$ are the covariance matrix and the coefficient matrix of the general active constraints, each with the columns of the fixed variables removed. $\boldsymbol{x'}$ is the vector of the remaining variables. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the adjustments due to the fixed variables:

$$\boldsymbol{\alpha} = \boldsymbol{C}\,\boldsymbol{x_f}\ , \quad \boldsymbol{\beta} = \boldsymbol{A_{g,a}}\,\boldsymbol{x_f}$$

$\boldsymbol{x_f}$ is the vector with $x_i = 0$ if no bound constraint is active for variable i, and $x_i = u_i$ or $x_i = l_i$, depending on whether the upper or lower bound is active.
At the beginning of an algorithm run, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ have to be calculated once. They then have to be updated each iteration if another bound becomes active or inactive. (If another general constraint changes status, the vectors on the right hand side of the system have to be expanded or shortened accordingly.)
The initial calculation of $\boldsymbol{\alpha}$ requires $O(\nu n)$ flops, the calculation of $\boldsymbol{\beta}$ amounts to $O(\nu\xi)$ flops. If the assumption of Subsection 3.3.1 – the number ν of active bounds is near n and the number ξ of general constraints is a (small) constant – still holds, the complexity of the initial calculation is $O(n^2)$. The adaptation of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to a changed active set requires $O(n)$ flops.
The main advantage gained by introducing this modification is a huge reduction of the size of matrix $\boldsymbol{H}$ (or $\hat{\boldsymbol{H}}$ respectively), and, as a consequence, a reduction of the number of flops needed to calculate and update the decomposition and also of the number of flops that are required to solve the system.
If there is a variable with identical upper and lower bound, then the value for this variable does not change at all during the computation of all segments. Therefore the variable

can – and should – be removed in a preprocessing step, due to the fact that only one of the bounds can be substituted into the system and the matrix $\boldsymbol{H}$ (or $\hat{\boldsymbol{H}}$) would then be singular. As a consequence, the corresponding element in the vector μ as well as the proper row and column in the covariance matrix have to be removed as well.
The presented technique does not depend on the order of the columns in matrix $\boldsymbol{H}$ or $\hat{\boldsymbol{H}}$. Both modifications – column rearrangement and the removal of variables with active bounds – can thus be used together.

3.3.3 System Split

Like Perold's algorithm [Per84], the algorithm developed by Best and Kale [BK00] removes the variables with active bounds, but additionally, it separates the system of equations into two parts that can be solved sequentially.
The first part consists of all rows of the Karush-Kuhn-Tucker (KKT) system that do not contain a Lagrange multiplier for either upper or lower bound, i.e. the matrix $\boldsymbol{A}^{\boldsymbol{T}}_{\boldsymbol{b,a}}$ has only zeroes in these rows.
Best and Kale called this system the *kernel system*, and the coefficient matrix on the left the *kernel matrix*:

$$\begin{pmatrix} \boldsymbol{C'_K} & \boldsymbol{A^T_{g,a,K}} \\ \boldsymbol{A'_{g,a}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{\lambda_g} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} + \boldsymbol{\alpha_K} \\ \boldsymbol{b_{g,a}} + \boldsymbol{\beta} \end{pmatrix} + \lambda_e \begin{pmatrix} \boldsymbol{\mu_K} \\ \boldsymbol{0} \end{pmatrix} \tag{3.14}$$

The index K denotes rows without any Lagrange multipliers for a single asset bound, the index $\overline{K}$ marks the rows that at least contain one Lagrange multiplier for a bound constraint.
The remaining lines of the KKT system form the second system of linear equations:

$$\begin{pmatrix} \boldsymbol{C'_{\overline{K}}} & \boldsymbol{A^T_{g,a,\overline{K}}} & \boldsymbol{A^T_{b,a,\overline{K}}} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{\lambda_g} \\ \boldsymbol{\lambda_b} \end{pmatrix} = \left(\boldsymbol{0} + \boldsymbol{\alpha_{\overline{K}}}\right) + \lambda_e \left(\boldsymbol{\mu_{\overline{K}}}\right) \tag{3.15}$$

By splitting the system of equations and removing the now unnecessary matrix $\boldsymbol{A}^{\boldsymbol{T}}_{\boldsymbol{b,a}}$ from the Kernel system, the decomposition of the Kernel matrix and the solution of this system require only a small fraction of the time that would be necessary otherwise.
If the assumption holds that only $n - \nu = c$ variables are neither at their upper nor at their lower bound and that no variable is both at its upper and lower bound, the kernel matrix has just the size $(c + \xi) \times (c + \xi)$.
Hence the time for the initial decomposition is not dependent on the number of investment alternatives n.
As soon as $\boldsymbol{x'}$ and $\boldsymbol{\lambda_g}$ are known, the computation of the Lagrange multipliers for the bounds is trivial, as each remaining multiplier appears in a separate row. $\boldsymbol{A}^{\boldsymbol{T}}_{\boldsymbol{b,a,\overline{K}}}$ is a permutation of the identity matrix. Due to its structure, it is not necessary to even store this matrix any more. It is sufficient to know which Lagrange multiplier appears in which row.

The update process at the end of an iteration varies depending on whether a general inequality or a bound constraint changes status:

- If a general inequality becomes active, it is added to the kernel matrix, i.e. an additional row is inserted into matrix $\boldsymbol{A'_{g,a}}$ and a column is inserted into $\boldsymbol{A^T_{g,a,K}}$. The right hand side of the new active constraint has to be modified to compensate for fixed variables due to active upper and lower bounds.
  Additionally, System 3.15 has to be adapted by including the correct column elements of the constraint gradient into matrix $\boldsymbol{A^T_{g,a,\overline{K}}}$.
  In case of a change from active to inactive, the opposite adaptations are required.

- If a bound constraint becomes active, the column associated with the variable that is now fixed is removed from the kernel matrix and from System 3.15. As a consequence, the right hand sides ($\boldsymbol{\alpha_K}$, $\boldsymbol{\alpha_{\overline{K}}}$, and $\boldsymbol{\beta}$) of both systems have to be modified as described.
  The new active upper or lower bound requires an associated Lagrange multiplier which entails that the row in which this multiplier appears has to be shifted from the kernel system to System 3.15.
  Should a bound constraint become inactive, the opposite modifications have to be performed.

It cannot be emphasized enough that, as a consequence of (i) the substitution of variables by their active bounds and (ii) of the additional splitting of the system of linear equations into two separately solvable parts, the matrix at the core of the each iteration (i.e. the kernel matrix) is normally tiny in comparison to the covariance matrix or the matrix $\boldsymbol{H}$ as defined in Section 3.1
Techniques to transform the system in a way to make the matrices more sparse, and then to use a sparse matrix decomposition (see e.g. Perold [Per84] or Jacobs et al. [JLM05]) are therefore neither necessary nor will they shorten computation time. Hence the origin and underlying structure of the covariance matrix is – as long as it is positive semidefinite – mostly irrelevant for the performance of this algorithm variant.
The presented algorithm also allows for an easy integration of linear transaction costs, or even piecewise linear transaction costs, as long as the cost function is convex. Such transaction costs change the function for the expected portfolio return from linear to piecewise linear and concave. Additional calculations have to be made only when a bound constraint is active, as now the multipliers of two adjacent constraints/segments have to be checked whether they become 0. For further details on how this can be realized, the reader is referred to Perold [Per84] or if a more detailed treatment is required, to Best and Hlouskova [BH05].

3.4 Test Results

The following test results were achieved on a problem with 225 investment alternatives from a benchmark set that was first used by Chang et al. [CMBS00] and is available

as part of the OR-Library [Bea06] and on three covariance matrices for 500, 1000, and 2000 shares kindly provided by the authors of Hirschberger et al. [HQS07]. They were generated by a procedure presented in that paper. The values for the expected returns were generated randomly for these three problems.
The computation time was measured on a computer with an AMD-Athlon-CPU with 1,5 GHz, the operating system we tested on was Linux. The program was compiled with version 3.3 of the Gnu Compiler Collection [GCC06] with the optimization level set to 3. Only the actual CPU-time used to calculate the solution was measured, as the complete runtime depends on the load of the operating system during computation. For the solution of the linear programming problems – both the maximization of the expected return and solution of **MIN-**λ_e – we used a commercial library, namely CPLEX 9.0 from ILOG [CPL06].
The listed values for the CPU-time are the averages of ten algorithm runs together with the associated standard error. The lower bound for all variables was set to 0, the upper bound to 0.1. No additional (general) constraints besides the budget constraint were used.
Table 3.1 shows the results for the different algorithm variants.

Table 3.1: Average CPU-time of PQP solver variants in seconds

Problem (problem size)	**P5** (225)	**P6** (500)	**P7** (1000)	**P8** (2000)
standard QR decomposition	**8.34**	**90.76**	**637.8**	**4137.5**
standard error	±0.005	±0.11	±0.98	±5.45
column rearrangement	**3.96**	**41.80**	**179.9**	**1453.0**
standard error	±0.014	±0.10	±0.93	±0.97
substitution of variables	**0.58**	**5.71**	**22.94**	**187.70**
standard error	±0.005	±0.03	±0.06	±0.48
system split variant	**0.09**	**0.48**	**1.39**	**6.75**
standard error	±0.004	±0.01	±0.01	±0.01

As can be expected, each improvement reduces computation time, with the system split variant being the fastest implementation by a wide margin. The reader may have noticed that this test setting, due to the fact that there are no constraints besides a single equality (budget constraints) and the bound constraints on the variables, is the most favorable for the system split algorithm.
But even if a small number of additional (general) constraints is introduced, the runtime of the system split algorithm does not increase dramatically. A comparison of the effect when 20 arbitrary "general" inequalities are included is shown in Table 3.2 for the problems with 500, 1000 and 2000 assets.

These general inequalities were constructed as follows:

$$\sum_{i \equiv 0 \pmod 2} x_i \leq 0.6 \tag{3.16a}$$

$$\sum_{i \equiv 1 \pmod 2} x_i \leq 0.6 \tag{3.16b}$$

$$\sum_{i \equiv k \pmod 3} x_i \leq 0.5 \quad k = 0, 1, 2 \tag{3.16c}$$

$$\sum_{i \equiv k \pmod 5} x_i \leq 0.4 \quad k = 0, 1, 2, 3 \tag{3.16d}$$

$$\sum_{i \equiv 4 \pmod 5} x_i \leq 0.35 \tag{3.16e}$$

$$\sum_{i \equiv 0 \pmod 7} x_i \leq 0.35 \tag{3.16f}$$

$$\sum_{i \equiv 1 \pmod 7} x_i \leq 0.33 \tag{3.16g}$$

$$\sum_{i \equiv k \pmod 7} x_i \leq 0.3 \quad k = 2, 3, 4, 5, 6 \tag{3.16h}$$

$$\sum_{i \equiv k \pmod{11}} x_i \leq 0.15 \quad k = 0, 1, 2 \tag{3.16i}$$

In order to increase the effect of these constraints, we set the upper bounds for all assets to 1, with the intention that many of the additional constraints should become active on large parts of the Pareto front.

Table 3.2: Average CPU-time per corner portfolio in seconds

Problem (problem size)	**P6** (500)	**P7** (1000)	**P8** (2000)
20 general inequalities	**0.00539**	**0.0130**	**0.0297**
standard error	±7.24e-6	±8.58e-6	±1.76e-5
no general inequalities	**0.00467**	**0.0118**	**0.0326**
standard error	±1.09e-5	±5.36e-6	±1.30e-5

Due to the inclusion of these constraints, the number of corner portfolios in the Pareto front changed. In order to get some meaningful results, we calculated the average CPU-time required to compute a corner portfolio.
The findings for the problems with 500 and 1000 assets are as expected: in both instances, there is a slight increase in computation time. As the last column shows, however, the obtained results have to be interpreted with great care: although the portfolio selection problem has now 20 additional general constraints, the average time required to calculate a corner portfolio was reduced for the problem with 2000 assets.

There are probably two reasons for this curious effect:

1. In the problem without general constraints, problem **MIN-**λ_e has to be solved, as the active constraints are linearly dependent. This was not the case when the additional constraints were included. The call to the LP-solver takes by far longer than the calculation of a "normal" segment.

2. In this special case at the end of the algorithm run, when λ_e was very small and variance reduction was more important than return maximization, the kernel matrix did get by far larger in the problem without additional constraints. This effect depended largely on the selection of the general constraints in this case. The results might be the other way round if either the problem data or coefficients in the inequalities were different.

Our tentative conclusion from this test is that the addition of a small number of general constraints to the problem does not seem to change the runtime of the system split variant very much. A more detailed analysis would require extensive further testing.

3.5 Implementation Details

The transformation of the algorithm framework presented in the previous section into a functional and efficient implementation is not straightforward. In this section, we will present two different matrix representations that were used to implement the system split variant, and we will explain the reason behind their selection for specific tasks in our algorithm. We will also highlight a small but critical part of the algorithm that is prone to numerical errors and a method to prevent them.

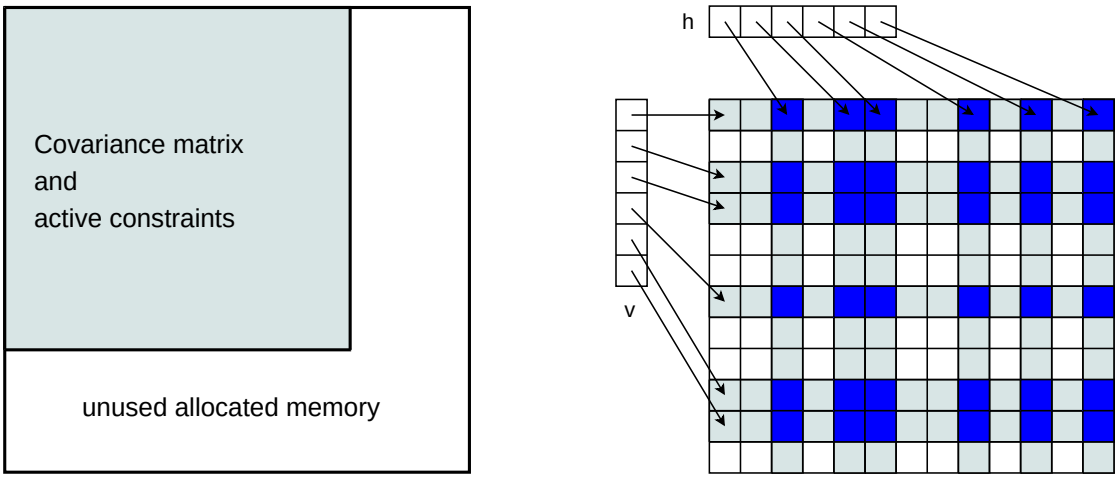
3.5.1 Matrix Representation

In the process of implementing the algorithm variants in C++, two different matrix representations were used, both intended for the storage of dense matrices.
The first representation is straightforward: the elements are stored, as is customary in C or C++, in row-major order. The matrix has, however, enough spare memory to handle the insertion of a predetermined number of rows and columns without having to allocate any more. Nevertheless, insertion and deletion of rows and columns – which are the operations that have to be performed in the presented algorithms – require extensive copy operations except if they are performed at one of the edges of the occupied area that adjoins the unused memory. Therefore the position of the matrix elements in the allocated space has to be chosen to fit these operations: if columns have to be inserted mainly at the right hand side of the matrix, it makes sense to store the matrix elements at the left edge of the allocated space (see e.g. Figure 3.1(a)).
With row-major storage structure, the unused memory to the right (or left) of the matrix is especially important. If the unused memory is placed to the right of the matrix, the number of necessary copy operations for column insertion at position k is reduced from

nearly n^2 (if there is no spare memory) to $(n-k)n$ (or to kn, if the free memory is placed to the left of the matrix).
The spare rows at the bottom (or top) don't influence the number of copy operations. They just allow insertion without dynamically allocating new memory. (It is just the other way round if a column-major storage scheme is used.)
Even more important, however, is that this representation stores all elements of a matrix row contiguously. If an appropriate container (e.g. a C-type array or the STL-vector in C++) is used, this storage layout allows for easy usage of so called *Basic Linear Algebra Subroutines*, a collection of highly efficient routines with a standardized interface that provide the building blocks for many linear algebra tasks. There are machine-optimized versions of the BLAS for nearly all known architectures. For a full overview of the standard and available implementations the reader is referred to the BLAS project page [BLA06]. Should there be no optimized version of the BLAS for a given environment, the developer can use ATLAS [ATL06], a program that generates an optimized BLAS-library for a given architecture by empirically tuning it.
But even if the linear algebra operations are implemented from scratch, the contiguous storage of the elements in one matrix row helps to shorten computation time by reducing the number of cache misses – assuming that matrix access is predominantly row-based.
If a matrix library does not offer this kind of representation directly, the same effect can usually be achieved by defining what is often called a *matrix view* on a given matrix, i.e. by creating a temporary object which gives access to a rectangular subset of the matrix. In case of insertion or deletion of a row or column, the copy operations are performed and then the old matrix view is deleted and a new view with modified size is created. In the remainder of this paper, we will call this matrix representation *submatrix representation.*

To cope with the main drawback of extensive copy operations of the submatrix representation, another matrix representation which allows cheap insertion and removal of both rows and columns was used. It works by channeling access to matrix elements through two *mapping vectors*, one to map vertical access ($\boldsymbol{v}$) and one to map horizontal access ($\boldsymbol{h}$). This is depicted in Figure 3.1b.
A prerequisite for the efficient application of this technique is that all possible matrix rows and columns are known from the start, and the actual matrix is just a subset of these rows and columns[2]. To prevent confusion of the reader, the matrix that contains all rows and columns will be called the *underlying matrix*, and the matrix that contains only the subsets of rows and columns will be referred to as *current matrix.* The two possible implementation alternatives are for the mapping vectors to be – and to remain – ordered or not; we decided on using sorted vectors to maintain the relative order of rows and columns.
If element (i, j) of the current matrix has to be accessed, this representation returns the element $(\boldsymbol{v_i}, \boldsymbol{h_j})$ of the underlying matrix.

[2]This is fulfilled for the algorithm variants that were presented above.



(a) Typical matrix representation with spare memory to make insertion easy

(b) Matrix representation with horizontal and vertical mapping vectors which allow easy insertion or deletion of rows and columns

Figure 3.1: Useful matrix representations for the algorithm implementation

Example:
If $\boldsymbol{v'} = (1, 3)$ and $\boldsymbol{h'} = (2, 3)$ and the underlying matrix looks as follows:

$$\begin{pmatrix} 1 & 4 & 7 & 8 \\ 2 & 6 & 4 & 0 \\ 3 & 5 & 2 & 6 \\ 0 & 8 & 4 & 7 \end{pmatrix}$$

then – if the indices are counted beginning with 1 – the current matrix is $\left(\begin{smallmatrix} 4 & 7 \\ 5 & 2 \end{smallmatrix}\right)$.
Should it be necessary to insert the first column of the underlying matrix, $\boldsymbol{h'}$ would be changed to $(1, 2, 3)$ and the current matrix would become $\left(\begin{smallmatrix} 1 & 4 & 7 \\ 3 & 5 & 2 \end{smallmatrix}\right)$.
Unfortunately, the elements that comprise one row of the current matrix are usually not stored contiguously, which – in case of large matrices – can lead to an increase in cache misses. This drawback also prevents the effective usage of standard BLAS-functions.
To show the consequence of using the mapping representation for row-based operations on medium- or large-scale matrices, we implemented the algorithm that uses column rearrangement both with the QR decomposition stored in a mapping representation and also in a submatrix representation. The results of the runtime tests are shown in Table 3.3. For both matrix representations, we used GSL-- [Chr06], an object-oriented wrapper for the GSL, and extended it with the required functionality; we did not use an architecture-optimized version of the GSL and its BLAS-operations. The runtime tests were performed on the same system as above, with 10 repetitions for each configuration and problem. Upper bounds for all assets were again set to 0.1, lower bounds remained at 0. No additional constraints beside the budget constraint were included.

Table 3.3: Comparison of the algorithm runtime (measured in CPU-seconds), using two different matrix representations for the QR decomposition

Problem (problem size)	**P5** (225)	**P6** (500)	**P7** (1000)	**P8** (2000)
column rearrangement (mapping)	**5.05**	**54.60**	**237.9**	**2467.9**
standard error	±0.004	±0.15	±0.21	±5.02
column rearrangement (submatrix)	**3.96**	**41.80**	**179.9**	**1453.0**
standard error	±0.014	±0.10	±0.93	±0.97

As can be seen clearly, the advantage of the submatrix representation is significant and increases with the problem size, even though no architecture-optimized BLAS-operations were used. If we had done that, the difference would have probably become even more pronounced.

We decided to incorporate both matrix representations into the algorithm in such a way as to maintain their respective advantages and reduce the drawbacks of each:
The matrix $\begin{pmatrix} \boldsymbol{C} & \boldsymbol{A}_g^T \\ \boldsymbol{A}_g & \boldsymbol{0} \end{pmatrix}$ contains all possible rows and columns of the kernel matrix (Eq. 3.14) and therefore also all rows and columns of the matrix $\begin{pmatrix} \boldsymbol{C}'_{\overline{K}} & \boldsymbol{A}_{g,a,\overline{K}}^T \end{pmatrix}$ which will be called *non-kernel-row matrix*.
Additionally, those general constraints which are inactive are also contained.
This clearly implies that it makes sense to use mapping vectors to manage all three matrices with the help of the mapping representation from above:

- The matrix $\begin{pmatrix} \boldsymbol{C} & \boldsymbol{A}_g^T \\ \boldsymbol{A}_g & \boldsymbol{0} \end{pmatrix}$ is used as *underlying matrix*.
- Three *current matrices* with different vertical mapping vectors but identical horizontal mapping are constructed:
  1. the kernel matrix
  2. the non-kernel-row matrix
  3. the matrix containing all inactive general constraints
- At any given moment in an algorithm run, every row can belong to one and only one vertical mapping vector. The operations that are necessary at the end of each segment include the manipulation of these vectors: when a row "leaves" one matrix, it is added to another.
- The horizontal mapping contains (ordered) references to the columns of all assets that are neither at their upper nor at their lower bound and to all columns with the gradients of the active general constraints. Should it be necessary, the horizontal mapping vector is updated at the end of segment as well.

This matrix representation permits insertion or deletion of row or column in a linear amount of time. For the storage of the QR decomposition, we used the submatrix

representation with unused space above and to the right of the matrix for both $\boldsymbol{Q^T}$ and $\boldsymbol{R}$ to allow for fast row operations.
For an extensive treatment of update and downdate operations in a QR factorization, the reader is referred to Golub and van Loan [GvL96, pp. 606-611], to Gill et al. [GMW91, pp. 133-138] and especially to Stewart [Ste98, pp. 326-353].
One might wonder if it is necessary to store the kernel matrix in addition to its decomposition. But due to the fact that the covariance matrix is normally stored in the main memory anyway, and the general constraints have to be stored somewhere as well, the explicit storage of the kernel matrix does not require a lot of additional memory (only $p(p+n)$ floating point numbers, if p is the number of general constraints), and it allows for an easier algorithm implementation, and – in the rare case this should happen when using a QR decomposition – it permits to check if numerical problems occurred with the decomposition.
Should main memory be scarce, it is possible, due to the symmetry of $\begin{pmatrix} \boldsymbol{C} & \boldsymbol{A_g^T} \\ \boldsymbol{A_g} & \boldsymbol{0} \end{pmatrix}$, to halve the required memory for the underlying matrix by overloading the access operator to return element (j,i) if element (i,j) is needed if it is situated in the nonexisting half of the matrix. (This was not implemented, however, as it would certainly increase the average time required to access an element, and memory was sufficiently available for the problem sizes we calculated. But for problems with 3000, 4000, or even more assets, it may turn out to be necessary.)

3.5.2 Cycling

The term *cycling* is used in the field of linear programming to describe a situation when the Simplex algorithm gets "stuck": the same vertex is reached again after two or more iterations without having made any progress with respect to the objective function.
An equivalent situation can occur during an algorithm run of the PQP algorithm: due to small numerical errors it is possible that one constraint becomes active in one iteration and inactive in the subsequent iteration without any (significant) change in λ_e. This could in the worst case lead to a stalling of the algorithm and, as a consequence, to possible memory overflow due to infinite cycling. In practice, a complete standstill happens only very rarely. The most probable reason for this fortunate behavior is that although the same constraints are active after two iterations in such a cycle, the QR decomposition varies slightly – due to small numerical errors as well.
For larger problem sizes, a repetition for several iterations can be observed in nearly every algorithm run, however. As each unnecessary iteration of the algorithm costs computation time and to prevent the rare event of a complete stalling, two modifications to the algorithm were tested:
Our first idea to prevent algorithm cycling was to modify Eq. 3.7 so that $\tilde{\lambda}_e$ is at least ϵ smaller than the current λ_e, where ϵ denotes a very small constant. This does eliminate almost all the cycles.
A suitable determination of ϵ is not easy, as λ_e does often comprise several orders of magnitude and ϵ must be large enough for the algorithm to " jump" over the numerical error. We got the best results when in each iteration epsilon was set as the product of the current λ_e and a small constant.

The biggest disadvantage of this approach is the rare case that another constraint would have become active in the interval $(\lambda_e - \epsilon, \lambda_e)$. (This is also the reason why ϵ should not be made too large.) If this happens, the Lagrange multiplier for the constraint that was overlooked becomes negative and optimality is not guaranteed any longer. To prevent this, additional checks would have to be implemented and the corner portfolio with negative Lagrange multipliers would have to be repaired. The repair however, has to be performed in a way that ensures that cycling does not occur in this part of the algorithm.

To sidestep this fairly complicated procedure, our second approach was to prevent an active constraint from becoming inactive if

1. it was made active in the last iteration and
2. the prospective $\tilde{\lambda}_e$ calculated due to this constraint is very nearly identical with the current λ_e.

As this approach blocks only one specific constraint, the probability of a wrong decision is by far smaller than in our first approach, where all constraints where prevented from becoming inactive in the interval $(\lambda_e - \epsilon, \lambda_e)$.
This approach does seem to eliminated cycling completely in our test problems. One can certainly construct a case in which cycling is not caused by only one constraint, but instead by a series of constraints that become inactive and active in a very unfortunate order without any change in λ_e. A situation like that – called degeneracy in the context of linear programming – does seem to happen more often when the objective function is linear. Elaborate strategies have been developed to cope with it (see e.g. Gill et al. [GMSW89]). The occurrence of such an event seems to be extremely rare with quadratic objective functions and, to our knowledge, has not occurred in several thousand different algorithm runs we have made.

3.6 Complete Algorithm Description

This section contains an in-depth description on how the modified version of the system split algorithm can be implemented when both the mapping and the submatrix representation and the mapping representation are used. It is our intention to support the reader in reimplementing it without too much difficulty.
The algorithm solves the following optimization problem for all $\lambda_e \in [0, \infty)$:

$$\min\{\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} - \lambda_e\boldsymbol{\mu}^T\boldsymbol{x} \mid \boldsymbol{A_g}\boldsymbol{x} \leq \boldsymbol{b_g},\ \boldsymbol{A_E}\boldsymbol{x} = \boldsymbol{b_E},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\} \tag{3.17}$$

where $\boldsymbol{u}$ and $\boldsymbol{l}$ are the upper and lower bounds, $\boldsymbol{A_g}\boldsymbol{x} \leq \boldsymbol{b_g}$ is the system of "general" inequalities, and $\boldsymbol{A_E}\boldsymbol{x} = \boldsymbol{b_E}$ is the system of equations.

The algorithm in detail:

1. Initialization:
   a) Calculate initial starting solution $\boldsymbol{x_0}$ by solving the optimization problem:

$$\min\{\boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} \mid \boldsymbol{A_g}\boldsymbol{x} \leq \boldsymbol{b_g},\ \boldsymbol{A_E}\boldsymbol{x} = \boldsymbol{b_E},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\} \tag{3.18}$$

   (Any reliable and fast LP-Solver will suffice, as the optimization problem is quite small for a linear program. The tolerances of the LP-Solver have to be chosen so that the results are sufficiently precise, however.)
   b) Determine at the point $\boldsymbol{x_0}$ the associated sets $I^0_{g,a}$ of active general constraints, $I^0_{l,a}$ of active lower bounds and $I^0_{u,a}$ of active upper bounds.
   c) Substitute the active bounds into the KKT-system by modifying the vector on the right hand side that is nonparametric with respect to λ_e:

$$\begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b_g} \end{pmatrix} \longrightarrow \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{b_g} + \boldsymbol{\beta} \end{pmatrix}$$

   with $\boldsymbol{\alpha} = \boldsymbol{C}\,\boldsymbol{x_f}$ and $\boldsymbol{\beta} = \boldsymbol{A_g}\,\boldsymbol{x_f}$ where $x_{f,i} = 0$ if neither upper nor lower bound of asset i is active, $x_{f,i} = u_i$ if asset i is at its upper bound, or $x_{f,i} = l_i$ if the lower bound of asset i is active.
   d) Generate two systems of linear equations and one system of inequalities based on the mapping representation with $\left(\begin{smallmatrix} \boldsymbol{C} & \boldsymbol{A_g^T} \\ \boldsymbol{A_g} & \boldsymbol{0} \end{smallmatrix}\right)$ as underlying matrix:
      i. the kernel system:

$$\begin{pmatrix} \boldsymbol{C'_K} & \boldsymbol{A^T_{g,a,K}} \\ \boldsymbol{A'_{g,a}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{\lambda_g} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\alpha_K} \\ \boldsymbol{b_{g,a}} + \boldsymbol{\beta_a} \end{pmatrix} + \lambda_e \begin{pmatrix} \boldsymbol{\mu_K} \\ \boldsymbol{0} \end{pmatrix}$$

      ii. the system containing the non-kernel-rows:

$$\begin{pmatrix} \boldsymbol{C'_{\overline{K}}} & \boldsymbol{A^T_{g,a,\overline{K}}} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{\lambda_g} \end{pmatrix} + \boldsymbol{A^T_{b,a,\overline{K}}}\,\boldsymbol{\lambda_b} = \boldsymbol{\alpha_{\overline{K}}} + \lambda_e\,\boldsymbol{\mu_{\overline{K}}}$$

      iii. the system of inactive general inequalities:

$$\begin{pmatrix} \boldsymbol{A'_{g,na}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{\lambda_g} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b_{g,na}} + \boldsymbol{\beta_{na}} \end{pmatrix}$$

   where the index K indicates rows that are part of the kernel systen, i.e. those rows that do not contain multipliers for active bounds, while in rows marked with the index $\overline{K}$, those multipliers do appear. Matrices marked with a prime are obtained by removing those columns that belong to variables whose upper or lower bound is active. Similarly, $\boldsymbol{x'}$ is the vector of all those asset weights that are not fixed. The index na denotes rows with inactive general inequalities.

The leftmost matrices in all three systems are constructed by using the underlying matrix from above together with the same horizontal mapping vector. This vector is composed of all i with $i \notin I_{l,a}^0$ and $i \notin I_{u,a}^0$ and of the index numbers of all active general constraints (both equations and inequalities) according to their position in the underlying matrix. (The vertical mapping vectors for the three matrices are disjoint.)

e) Generate a QR decomposition of the kernel matrix $\begin{pmatrix} \boldsymbol{C'_K} & \boldsymbol{A^T_{g,a,K}} \\ \boldsymbol{A'_{g,a}} & \boldsymbol{0} \end{pmatrix}$. The matrices $\boldsymbol{Q^T}$ and $\boldsymbol{R}$ are stored using the submatrix representation[3]. If m is the number of general constraints (equations and inequalities), memory of the size of $(n+m)^2$ floating point numbers is allocated for each of the matrices[4]. The elements of both matrices are stored in the bottom left corner of the allocated memory.

f) Set $k \longleftarrow 0$ and set $\lambda_e^k \longleftarrow \infty$.

2. If $\begin{pmatrix} \boldsymbol{C'_K} & \boldsymbol{A^T_{g,a,K}} \\ \boldsymbol{A'_{g,a}} & \boldsymbol{0} \end{pmatrix}$ is nonsingular (which can be checked easily by examining if the absolute value of all diagonal elements in $\boldsymbol{R}$ are larger than a threshold ϵ) then

a) Calculate the vectors $\begin{pmatrix} \boldsymbol{h_{x',q}} \\ \boldsymbol{h_{\lambda_g,q}} \end{pmatrix}$ and $\begin{pmatrix} \boldsymbol{h_{x',p}} \\ \boldsymbol{h_{\lambda_g,p}} \end{pmatrix}$ which determine $\boldsymbol{x'}$ and $\boldsymbol{\lambda_g}$ in the current segment:

$$\begin{pmatrix} \boldsymbol{x'} \\ \boldsymbol{\lambda_g} \end{pmatrix} = \begin{pmatrix} \boldsymbol{h_{x',q}} \\ \boldsymbol{h_{\lambda_g,q}} \end{pmatrix} + \lambda_e \begin{pmatrix} \boldsymbol{h_{x',p}} \\ \boldsymbol{h_{\lambda_g,p}} \end{pmatrix}$$

The vectors are calculated by solving the following two systems of equations using back substitution:

$$\boldsymbol{R} \begin{pmatrix} \boldsymbol{h_{x',q}} \\ \boldsymbol{h_{\lambda_g,q}} \end{pmatrix} = \boldsymbol{Q^T} \begin{pmatrix} \boldsymbol{\alpha_K} \\ \boldsymbol{b_{g,a}} + \boldsymbol{\beta_a} \end{pmatrix}$$

$$\boldsymbol{R} \begin{pmatrix} \boldsymbol{h_{x',p}} \\ \boldsymbol{h_{\lambda_g,p}} \end{pmatrix} = \boldsymbol{Q^T} \begin{pmatrix} \boldsymbol{\mu_K} \\ \boldsymbol{0} \end{pmatrix}$$

b) Calculate the vectors $\boldsymbol{h_{\lambda_b,q}}$ and $\boldsymbol{h_{\lambda_b,p}}$ as follows[5]:

$$\boldsymbol{h_{\lambda_b,q}} = \boldsymbol{\alpha_{\overline{K}}} - \begin{pmatrix} \boldsymbol{C'_{\overline{K}}} & \boldsymbol{A^T_{g,a,\overline{K}}} \end{pmatrix} \begin{pmatrix} \boldsymbol{h_{x',q}} \\ \boldsymbol{h_{\lambda_g,q}} \end{pmatrix}$$

$$\boldsymbol{h_{\lambda_b,p}} = \boldsymbol{\mu_{\overline{K}}} - \begin{pmatrix} \boldsymbol{C'_{\overline{K}}} & \boldsymbol{A^T_{g,a,\overline{K}}} \end{pmatrix} \begin{pmatrix} \boldsymbol{h_{x',p}} \\ \boldsymbol{h_{\lambda_g,p}} \end{pmatrix}$$

[3] The matrix $\boldsymbol{Q^T}$ was used instead of $\boldsymbol{Q}$, as only the transposed is required in the following calculations.

[4] If memory is scarce, it may be necessary to handle the allocation more dynamically, as usually the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are much smaller than $(n+m) \times (n+m)$.

[5] This works only if the matrix $\boldsymbol{A^T_{b,a,\overline{K}}}$ is the identity matrix. Otherwise, i.e. if $\boldsymbol{A^T_{b,a,\overline{K}}}$ is a permutation matrix, the order of the elements has to be changed accordingly.

c) Determine the values $\hat{\lambda}_{e,g}$, $\hat{\lambda}^u_{e,b}$, $\hat{\lambda}^l_{e,b}$, $\tilde{\lambda}_{e,g}$, and $\tilde{\lambda}_{e,b}$:

$$\hat{\lambda}_{e,g} = \max \left\{ \frac{b_{g,na,i} + \beta_{na,i} - (\boldsymbol{a'_{g,na,i}})^T \boldsymbol{h_{x',q}}}{(\boldsymbol{a'_{g,na,i}})^T \boldsymbol{h_{x',p}}} \right.$$
$$\left. \forall\, i = 1, \ldots, m \text{ with } i \notin I_{g,a} \text{ and } \boldsymbol{a_i}^T \boldsymbol{h_{x,p}} < -\epsilon \right\}$$

$$\hat{\lambda}^u_{e,b} = \max \left\{ \frac{u'_i - h_{x',q,i}}{h_{x',p,i}} \;\forall\, i = 1, \ldots, n \text{ with } i \notin I_{u,a} \cup I_{l,a} \text{ and } h_{x',p,i} < -\epsilon \right\}$$

$$\hat{\lambda}^l_{e,b} = \max \left\{ \frac{l'_i - h_{x',q,i}}{h_{x',p,i}} \;\forall\, i = 1, \ldots, n \text{ with } i \notin I_{u,a} \cup I_{l,a} \text{ and } h_{x',p,i} > \epsilon \right\}$$

$$\tilde{\lambda}_{e,g} = \max \left\{ -\frac{h_{\lambda_g,q,i}}{h_{\lambda_g,p,i}} \;\forall\, i = 1, \ldots, m \text{ with } i \in I_{g,a} \text{ and } h_{\lambda_g,p,i} > \epsilon \right\}$$

$$\tilde{\lambda}_{e,b} = \max \left\{ -\frac{h_{\lambda_b,q,i}}{h_{\lambda_b,p,i}} \;\forall\, i = 1, \ldots, n \right.$$
$$\left. \text{with } i \in I_{u,a} \cup I_{l,a} \text{ and } h_{\lambda_b,q,i} h_{\lambda_b,p,i} < 0 \text{ and } -\frac{h_{\lambda_b,q,i}}{h_{\lambda_b,p,i}} < \lambda_e + \epsilon \right\}$$

d) Set $\lambda_e^{k+1} \longleftarrow \max\{\hat{\lambda}_{e,g}, \hat{\lambda}^u_{e,b}, \hat{\lambda}^l_{e,b}, \tilde{\lambda}_{e,g}, \tilde{\lambda}_{e,b}\}$

e) If $\lambda_e^{k+1} = \hat{\lambda}_{e,g}$ and $\hat{\lambda}_{e,g}$ is determined by index j,
- set $I_{g,a}^{k+1} \longleftarrow I_{g,a}^k \cup \{j\}$
- add row and column for general constraint j to the kernel matrix
- update QR decomposition

else if $\lambda_e^{k+1} = \tilde{\lambda}_{e,g}$ and $\tilde{\lambda}_{e,g}$ is determined by index j,
- set $I_{g,a}^{k+1} \longleftarrow I_{g,a}^k \setminus \{j\}$
- remove row and column for general constraint j from the kernel matrix
- downdate QR decomposition

else if $\lambda_e^{k+1} = \hat{\lambda}^u_{e,b}$ and $\hat{\lambda}^u_{e,b}$ is determined by index j
- set $I_{u,a}^{k+1} \longleftarrow I_{u,a}^k \cup \{j\}$
- substitute u_j into the underlying matrix which changes $\boldsymbol{\alpha_K}$, $\boldsymbol{\alpha_{\overline{K}}}$, $\boldsymbol{\beta_a}$, and $\boldsymbol{\beta_{na}}$ and then remove the column associated with asset j from the kernel matrix
- shift the row that will contain the multiplier of the new bound constraint from the kernel matrix to the non-kernel-row matrix
- downdate QR decomposition

else if $\lambda_e^{k+1} = \hat{\lambda}^l_{e,b}$ and $\hat{\lambda}^l_{e,b}$ is determined by index j
- set $I_{l,a}^{k+1} \longleftarrow I_{l,a}^k \cup \{j\}$
- substitute l_j into the underlying matrix which changes $\boldsymbol{\alpha_K}$, $\boldsymbol{\alpha_{\overline{K}}}$, $\boldsymbol{\beta_a}$, and $\boldsymbol{\beta_{na}}$ and then remove the column associated with asset j from the kernel matrix
- shift row that will contain the multiplier of the new bound constraint from the kernel matrix to the non-kernel-row matrix
- downdate QR decomposition

else if $\lambda_e^{k+1} = \tilde{\lambda}_{e,b}$ and $\tilde{\lambda}_{e,b}$ is determined by index j

- if $j \in I_{l,a}^k$: set $I_{l,a}^{k+1} \longleftarrow I_{l,a}^k \setminus \{j\}$ and change back $\boldsymbol{\alpha_K}$, $\boldsymbol{\alpha_{\overline{K}}}$, $\boldsymbol{\beta_a}$, and $\boldsymbol{\beta_{na}}$ by "unsubstituting" l_j
- if $j \in I_{u,a}^k$: set $I_{u,a}^{k+1} \longleftarrow I_{u,a}^k \setminus \{j\}$ and change back $\boldsymbol{\alpha_K}$, $\boldsymbol{\alpha_{\overline{K}}}$, $\boldsymbol{\beta_a}$, and $\boldsymbol{\beta_{na}}$ by "unsubstituting" u_j
- Insert the column associated with asset j into the kernel matrix
- shift row that contains the multiplier which has become 0 from the non-kernel-row matrix to the kernel matrix
- downdate QR decomposition

f) Set $x_i^{k+1} \longleftarrow h_{x',q,i} + \lambda_e^{k+1} h_{x',p,i}$ for the variables that are not fixed, and set $x_i^{k+1} = u_i$ or $x_i^{k+1} = l_i$ if the upper or lower bound of asset i is active in the current segment.

Otherwise, i.e. the matrix $\begin{pmatrix} \boldsymbol{C'_K} & \boldsymbol{A^T_{g,a,K}} \\ \boldsymbol{A'_{g,a}} & \boldsymbol{0} \end{pmatrix}$ is singular:

a) Calculate λ_e^{k+1}, $\boldsymbol{\lambda_g}$, $\boldsymbol{\lambda_b}$, and $\boldsymbol{x^{k+1}}$ by solving the linear optimization problem **MIN-λ_e-l-u**:

$$\min \ \lambda_e^{k+1} \tag{3.19a}$$

$$\begin{pmatrix} \boldsymbol{C'_K} & \boldsymbol{A^T_{g,a,K}} \\ \boldsymbol{A'_{g,a}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'^{(k+1)}} \\ \boldsymbol{\lambda_g} \end{pmatrix} - \lambda_e^{k+1} \begin{pmatrix} \boldsymbol{\mu_K} \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\alpha_K} \\ \boldsymbol{b_{g,a}} + \boldsymbol{\beta_a} \end{pmatrix} \tag{3.19b}$$

$$\begin{pmatrix} \boldsymbol{C'_{\overline{K}}} & \boldsymbol{A^T_{g,a,\overline{K}}} \end{pmatrix} \begin{pmatrix} \boldsymbol{x'^{(k+1)}} \\ \boldsymbol{\lambda_g} \end{pmatrix} + \boldsymbol{A^T_{b,a,\overline{K}}}\, \boldsymbol{\lambda_b} = \boldsymbol{\alpha_{\overline{K}}} + \lambda_e^{k+1}\, \boldsymbol{\mu_{\overline{K}}} \tag{3.19c}$$

$$\boldsymbol{A_g}\, \boldsymbol{x^{k+1}} \leq \boldsymbol{b} \tag{3.19d}$$

$$\boldsymbol{l} \leq \boldsymbol{x^{k+1}} \leq \boldsymbol{u} \tag{3.19e}$$

$$\boldsymbol{\lambda_b}, \boldsymbol{\lambda_g} \geq \boldsymbol{0} \tag{3.19f}$$

$$\lambda_e \geq 0 \tag{3.19g}$$

with all variables defined as above.

b) Determine the active constraints that will leave $I_{g,a}^k$, $I_{l,a}^k$, and $I_{u,a}^k$ by checking if any of the multipliers in $\boldsymbol{\lambda_b}$ and $\boldsymbol{\lambda_g}$ have become 0 at the solution point. Calculate residuals (e.g. $\boldsymbol{r_g} = \boldsymbol{b} - \boldsymbol{A_g}\, \boldsymbol{x}$) for all inactive constraints to determine which of those will become active. Handle activation and deactivation of constraints as described in Step 2e of the algorithm.

3. If $\lambda_e^k > 0$, store the calculated solution $\boldsymbol{x^{k+1}}$ and go to step 2,
otherwise (i.e. if the end of the algorithm run is reached) calculate and store

$$\boldsymbol{x^{k+1}} \longleftarrow \frac{\lambda_e^k}{\lambda_e^k - \lambda_e^{k+1}}\, \boldsymbol{x^{k+1}} + \frac{\lambda_e^{k+1}}{\lambda_e^k - \lambda_e^{k+1}}\, \boldsymbol{x^k}$$

4. Terminate.

3.7 Summary and Concluding Remarks

This chapter details several variants of active set algorithms for portfolio selection and documents their runtimes on large problem instances.
In Section 3.1 we present an algorithm framework based on the work of Best [Bes96]. We have adapted this algorithm to portfolio selection problems and have extended it in a way that enables it to cope with several linearly dependent active constraints – even if such a situation should occur at the starting point of the algorithm. Section 3.2 gives an overview of related work and highlights the difference between the parametric quadratic programming algorithms for portfolio selection and "conventional" active-set algorithms.
Based on the characteristic of realistic portfolio selection problems to have both upper and lower bounds for all or nearly all assets, Section 3.3 describes how the presented algorithm framework can be modified to speed up the time required for the computation of the Efficient Frontier. We present three implementation variants: (i) column rearrangement, (ii) the substitution of upper and lower bounds and (iii) the separation of the system of linear equations into 2 parts that can be solved sequentially. In Section 3.4 the performance improvement of these implementation variants are documented on medium- to large-scale portfolio selection problems. The fastest algorithm, the system split variant, is able to solve large problem instances in a very short time and has proven to be suitable even if the number of general inequalities is increased.
Section 3.5 describes two matrix representations, the submatrix representation and mapping representation, and outlines how they can be used to efficiently implement the system split variant so that it benefits from the advantages of both representations without having to put up with most of their drawbacks. In this section we also describe a simple way to prevent algorithm stalling that is caused by small numerical errors. If not taken care of, this problem can occur quite regularly and can slow down the algorithm considerably.
Since, as far as we know, there is no implementation of a PQP solver in the public domain that is both efficient and able to cope with large problem instances, we provide an in-depth description of the fastest algorithm variant in Section 3.6. This should help other practitioners in easily reimplementing it so that they can use it in their own research without too much effort.

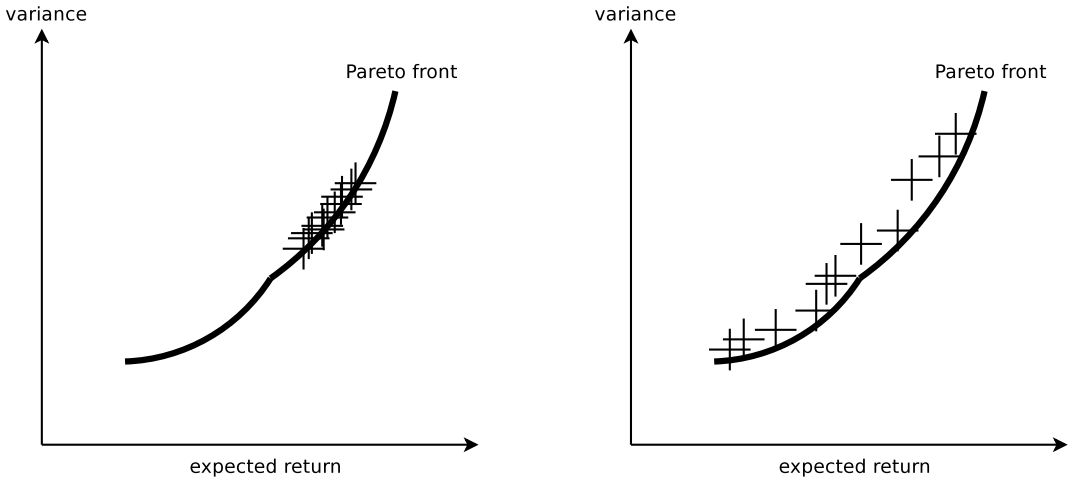
Extensions:

The ability to cope with piecewise linear transaction costs would broaden the possible applications of the algorithms, and we intend to include this functionality in the near future, as we think practitioners would find this especially helpful when using the presented PQP solver.
Another interesting, although possibly complicated, extension of our algorithm would be the integration of at least one additional linear optimization criterion.
With respect to the actual implementation, a relatively easy-to-do improvement would be the switch to architecture-optimized BLAS-codes. This should further shorten algorithm

run times, although we assume that the improvement would only be minor due to the small matrices in the system split variant. (Optimized BLAS routines show their advantages not until the matrices reach a certain size. See e.g. Mello and Khabibrakhmanov [MK06].) Nevertheless, an actual in-depth profiling of algorithm variants that use different matrix libraries with respect to occurring cache misses might yield interesting insights.

4 Point-Based Solution Approaches Based on the ϵ-Constraint Method

Point-based solution approaches for multicriteria problems try to approximate a Pareto front by generating a number of points that are close to the front. To achieve a good approximation of the complete front, proximity of all points to the Pareto front is not sufficient, however, as this does not guarantee to cover the front in its whole breadth: Figure 4.1(a) shows an (imaginary) approximation for a Pareto front with all its solutions very close to the actual front (which usually would not be known) — but they are all situated in a narrow interval. The heuristic that produced solution (a) might be the right choice just for those investors that already know their area of interest is in this interval. Figure 4.1(b) depicts a set of points that are distributed more evenly over the whole Pareto front, but the individual solution is on average farther away from the front. If the main goal is a reasonably good approximation of the complete Pareto front, the heuristic that calculated the solutions in (b) may be preferable to (a).



(a) All solutions in narrow interval, but close to the Pareto front

(b) Solutions distributed widely, significant distance between individual solution and Pareto front

Figure 4.1: Comparison of two approximations for a Pareto front in mean-variance space

A successful point-based approach that intends to approximate the complete Efficient Frontier has to perform well in both aspects: getting closer to the front and maintaining an even distribution.

These tasks can be tackled either sequentially – one part of the algorithm is responsible for the distribution of the points, another tries to place the calculated portfolios as close to the actual Pareto front as possible – or simultaneously – the algorithm is constructed in such a way that the two aspects are interwoven and can not be separated without a complete redesign.
Algorithms that handle both goals separately usually transform the bicriteria problem into an optimization problem with a single objective in the same way as demonstrated for the convex problem in Section 2.2.
Unlike in the convex case, however, only the approach to transform the second optimization criterion into an additional restriction (ϵ-Constraint method) can be applied here, since a linear combination of the two objectives can not be used to identify those regions of the Pareto front that are nonconvex (see e.g. Jobst et al. [JHLM01]).

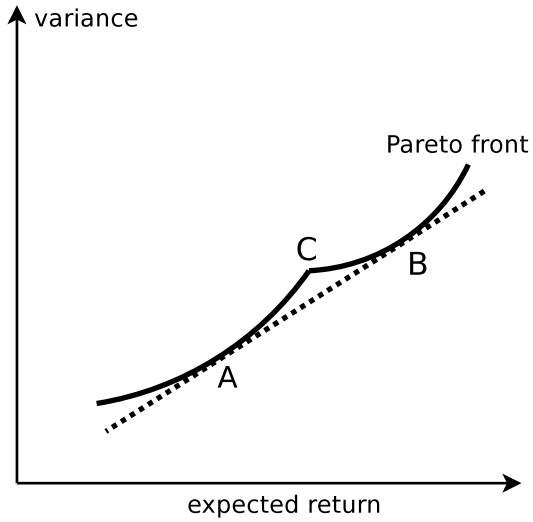


Figure 4.2: Inability to reach nonconvex sections of the Pareto front (region ABC) with the weighting approach.

Figure 4.2 illustrates this. Region ABC of the Pareto front can never be obtained by using an objective function $F(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} - \lambda_e\boldsymbol{x}^T\boldsymbol{\mu}$ regardless of the value for λ_e. As it is usually easier to work with a quadratic objective function and linear constraints instead of a linear objective function and a quadratic constraint, the expected return criterion is converted into another constraint:

$$\boldsymbol{x}^T\boldsymbol{\mu} \geq E_f$$

In this chapter we will focus on using the ϵ-Constraint method to generate point-based approximations for mean-variance portfolio selection problems that contain, among regular convex constraints, one of three nonconvex constraints: Either (i) a maximum cardinality constraint, or (ii) a minimum threshold constraint, or (iii) the 5-10-40-Constraint

based on German investment law. (Chapter 5 presents multi-objective evolutionary algorithms that do not require the transformation of one of the objectives into another constraint. They handle both optimization aspects simultaneously.)

The remainder of this chapter is structured as follows:
In Section 4.1, we start by describing how we measure solution quality for problems with nonconvex constraints. Section 4.2 presents a 2-phase algorithm that efficiently distributes the points over the Pareto front with the goal to get the best approximation when only a limited number of points (portfolios) can be calculated. In Section 4.3 we discuss whether the problem formulation of the three problem types (problems with either a maximum cardinality, buy-in thresholds, or the 5-10-40-Constraint) has to be modified in order to allow the application of commercially available high performance mixed-integer solvers, and if so, how the modifications should look like. Section 4.4 presents heuristics for each nontrivial constraint type that, given a lower bound for the expected return, aim at calculating the portfolio with the smallest variance. These heuristics have been tested on 7 benchmark problems, and the obtained results are presented and analyzed in Section 4.5. Additionally, in this section, we report on tests that give an approximation on the effectiveness of the 2-phase procedure for point distribution. We conclude this chapter with a brief summary in Section 4.6.

4.1 Performance Measurement

Measuring performance in a multi-objective setting is difficult, because it requires the comparison of complete frontiers or approximations thereof and not only singular solutions/portfolios. A number of possible performance measures are discussed e.g. in Hansen and Jaszkiewicz [HJ98] or in Zitzler et al. [ZTL$^+$03]. In the following, we will judge a generated front by its deviation from the *ideal front*, which is defined as the efficient frontier of the problem without non-convex constraints[1]. This ideal front is an upper bound on the performance and can be computed efficiently with parametric quadratic programming, like e.g. the algorithm presented in Chapter 3. To measure the deviation, we calculate the area between the resulting front and the ideal front. One difficulty with area-based methods is to define the maximum variance and minimum return boundaries to calculate the area, see Figure 4.3. If these values are set far apart, extreme portfolios have a very large impact on solution quality. If they are set too close, some parts of the front may be cut off. Since the appropriate borders are not clear, we report on two values here: The area using the maximum variance and minimum yield portfolios from the ideal front (ideal delta-area), and the maximum variance and minimum yield from any asset in the available universe (max. delta-area).
Figure 4.3 illustrates how the success of an optimization run is measured for point-based approximations. If an approximation for the Pareto front can be described not only as a collection of points, but as a set of curves – as e.g. depicted in Figure 4.4 – the optimization results are measured analogously. The only difference now is that the area to be calculated does not have a stair-shaped appearance any more. Chapters 5 and 6

[1] For the problem with 5-10-40-Constraint, the upper bound of all assets is set to 10%.

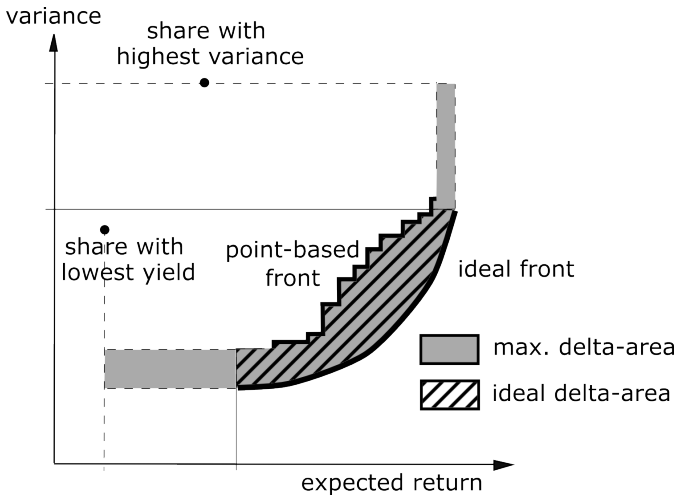


Figure 4.3: Ideal and maximum delta-area for point-based solution approaches.

present methods with this capability (envelope-based algorithms). An algorithm that reduces the negative effect of the "stairs" in Fig. 4.3 is presented in the following section.

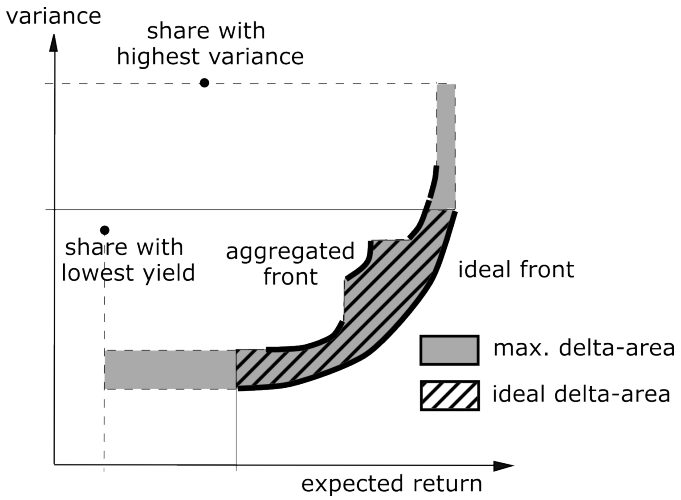


Figure 4.4: Ideal and maximum delta-area for envelope-based solution approaches.

4.2 Efficient Distribution of Points

When a limited number of points is used to approximate the Pareto front, there is always an approximation error due to the fact that nothing is known about the solutions between two adjacent points. This error exists even if all those points lie directly on the actual Pareto front.

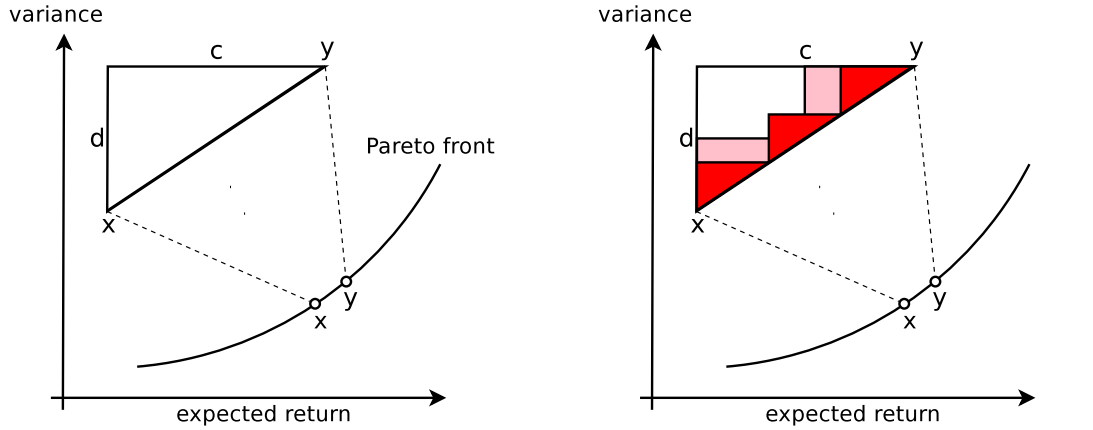
Figure 4.5(a) illustrates the problem: suppose that points x and y with the coordinates (x_e, x_v) and (y_e, y_v) on the Pareto front are known, but no other points in-between. If a prospective investor wants to achieve an expected return that is larger than x_e, she has no choice but to at least invest in Portfolio y. Thus, she has to accept the higher variance y_v that goes with it. If, however, the portfolio of an investor is required to have a variance below y_v, then x_e is the maximum expected return she can achieve.
As described in Section 4.1, we use the area between the solution of the relaxed problem and the approximation to measure algorithm success. The additional area which is the result of using a point-based approximation instead of the actual Pareto front in the interval (x_e, y_e) can be estimated as $F_{\text{ap}} = \frac{1}{2}cd$. Obviously, this can only be a very rough estimate as we do not know the shape of the actual Efficient Frontier between x and y. The shorter the interval (x_e, y_e), however, the more the Pareto front will usually resemble a straight line, and the more accurate the approximation will be.
If an infinite number of points could be calculated, the approximation error would converge to 0. In reality, only a limited amount of time is available, and therefore we can only calculate a limited number of points (portfolios) – which will be called the *computation budget* henceforth. We will now describe an algorithm that, given such a computation budget, determines where the points are to be placed on the expected return axis in order to reduce the approximation error as far as possible.
In order to be able to calculate the distribution of points, we will again assume that the Pareto front can be linearized piecewise: given a set of points (portfolios) that are already known, we assume that the points between two adjacent portfolios are on or near a direct line between those two points.
Figure 4.5(b) illustrates the improvements that can be achieved if one or two additional points are placed between x and y. (We again assume linearity of the Pareto front in the interval $[x_e, y_e]$.)
Given two points x and y on the Pareto front with the coordinates (x_e, x_v) and (y_e, y_v), $x_e < y_e$, the reduction of the approximation error that is achieved by inserting n evenly spaced additional points can be calculated as follows:

$$\Delta F_{\text{ap}} = mc^2 \frac{n}{2(n+1)}$$

where $c = y_e - x_e$ and $m = (y_v - x_v)/c$.
The additional reduction of the approximation error when we evenly distribute n points between x and y instead of $n-1$, is therefore:

$$\Delta F_{\text{ap}}^{(n-1)\to n} = mc^2 \frac{n}{2(n+1)} - mc^2 \frac{n-1}{2n} = \frac{mc^2}{2} \frac{1}{n(n+1)} \tag{4.1}$$

The basic idea of our algorithm is as follows:
In a first phase, a set of different values for the lower bound E_f of the expected return is determined by distributing a small share of the computation budget equidistantly over the interval $[E_{\min}, E_{\max}]$. The minimum yield $E_{\min}$ and the maximum yield $E_{\max}$ have to be chosen in a way that no part of the Pareto front is inadvertently cut off. Then,



(a) Estimate of the approximation error if no points between x and y are known

(b) Improvement of the approximation by calculating one or two additional solutions between x and y

Figure 4.5: Estimation error of point-based solutions

a solution for each of those values for E_f is calculated. In the second phase of our algorithm, based on the solutions from the first phase, new values for E_f are determined by means of Eq. 4.1 so that the approximation error is minimized. For each newly determined value of E_f, we also solve a single-criterion optimization problem with E_f as the lower bound for the expected return.
In the subsequent sections we will present several methods to solve these single-criterion problems. Important for the design of the distribution algorithm is not how they work in detail, but if they are capable of determining $E_{\min}$ and $E_{\max}$ exactly (mixed integer approaches, Section 4.3) or not (heuristic approaches, Section 4.4).

As simple as the basic concept for the algorithm seems, there are several aspects of it that have to be considered in more detail, some of which depend partially on whether a mixed integer solver or a heuristic is used to calculate the single criterion solutions.

Determination of $E_{\max}$ and $E_{\min}$

If a mixed integer solver for quadratic programming (MIQP solver) is available and the nonconvex problem can be modeled as a mixed integer problem (see Section 4.3), it is easy to get precise values for both $E_{\max}$ and $E_{\min}$. $E_{\max}$ can simply be determined by maximizing the expected return with respect to both convex and nonconvex constraints. (For this, even a mixed integer solver for linear problems is sufficient.) $E_{\min}$ is just the minimum variance portfolio that complies with both types of constraints. (This problem requires a mixed integer solver that can cope with a quadratic objective function.)
If we cannot use an MIQP solver – either because none is available or since the computation of the solution would need too much time – and we have to rely on heuristics, $E_{\max}$ and $E_{\min}$ have to be determined by other means. A reasonable value for $E_{\max}$ is

the maximum return of the ideal front, i.e. the highest return that can be achieved when the nonconvex constraint is not considered. Usually, this value is sufficiently close to the maximum yield of the nonconvex problem so that not much of the computation budget is wasted.
Unfortunately, $E_{\min}$ cannot be determined as easily and as precisely. The yield of the minimum variance portfolio (MVP) of the ideal front is too restricting, as quite often there are nondominated[2] solutions with an expected return below this value.
Two approaches to determine $E_{\min}$ seemed reasonable for us: The first one – which might be suitable for the practitioner – is to look at the share data, and, if available, at results of previous optimization runs and then to make an educated guess. The second approach is to calculate the minimum yield that can be achieved when all constraints beside the nonconvex constraint are considered. We did opt for the second method, since it allows to specify $E_{\min}$ without any external input, and we can be sure not to have cut off a part of the Pareto front. The main problem with the second approach is that the minimum expected return is usually far smaller than the yield of the portfolio at the low-variance boundary of the Pareto front. As a consequence, a good part of the computation budget used in the first phase is wasted beyond the left edge of the Efficient Frontier.
If an investor is only interested in a segment of the Pareto front, $E_{\min}$ and $E_{\max}$ can be chosen according to her preferences.

Border Areas of the Phase 1-Results

The problem of what is to be done after Phase 1 to the left of the leftmost nondominated portfolio and to the right of the respective rightmost nondominated portfolio depends again on wether an MIQP solver or some heuristic is used. If a mixed integer solver is available and has been used to determine $E_{\min}$ and $E_{\max}$, the portfolios that correspond to $E_{\min}$ and $E_{\max}$ are part of the Pareto front and mark the respective boundaries. No further measures are necessary, as the Pareto front spans the whole interval $[E_{\min}, E_{\max}]$.
In case heuristics are used, the Pareto front usually spans only part of the interval $[E_{\min}, E_{\max}]$. The two border areas require special handling:
If the lower bound for the expected return E_f is smaller than the expected return E_l of the portfolio that has the lowest variance of any portfolio the heuristic is able to calculate, this expected return constraint should not be binding. The heuristic will therefore return the same portfolio for all values of $E_f < E_l$[3]. Hence, the expected return of the nondominated portfolio with the lowest variance calculated in Phase 1 can be assumed to mark the minimum-variance edge of the Pareto front.
If E_r is the highest expected return value the heuristic could find a feasible solution for, and h_1 is the number of portfolios allocated to Phase 1, then we know that right

[2] A solution of a multicriteria optimization problem is nondominated with respect to a given set S of other solutions if and only if there is no other solution in S that is better in at least one criterion and not worse in all other criteria. Otherwise, the solution is dominated.
[3] It is certainly possible to imagine heuristics that are designed otherwise, but for the heuristics presented in this thesis, the statement can be accepted to be true.

edge of the Pareto front is somewhere in the interval $[E_r, E_r + \frac{E_{\max} - E_{\min}}{h_1 - 1})$. As we have no further information about the characteristics of the Pareto front in the interval, we decided to allocate a "neutral" share of the remaining budget on this interval, i.e. the interval receives roughly the number of points that it would get if the points would be distributed equally spaced in Phase 2 as well. The number of points r in the interval is calculated as follows:

$$r = \left\lfloor \frac{E_{\max} - E_{\min}}{(h-1)(E_r - E_l)} (t - h) \right\rfloor$$

where t is the computation budget, i.e. the number of portfolios that we are allowed to calculate in total.

Dividing the Computation Budget on Phase 1 and Phase 2

What share of our computation budget is invested in Phase 1 depends on

1. the total number of portfolios that can be calculated:
   If the total budget is very small, enough of it has to be invested into Phase 1 in order to at least get a small number of nondominated points as a result. If Phase 1 returns only one or two nondominated solutions, the advantages of our algorithm are unable to come into play.

2. whether a MIQP solver or heuristics are used to calculate the points:
   In case of the heuristics, a part of the budget in Phase 1 is "wasted" on the area below the minimum-variance-boundary of the Pareto front. Therefore the share of Phase 1 should be neither too small (may result in not enough nondominated solutions) nor too large (a larger part of the overall budget is wasted below the MVP). If an MIQP solver is used, the number of portfolios in Phase 1 is less relevant.

Concluding, the algorithm needs enough portfolios to get reasonable results in Phase 1 and has to have left enough of the budget for Phase 2 to take advantage of the information gathered in first phase. During our tests, we used an overall budget of several hundred points, and we decided on allocating 10% of it to Phase 1, we used a small number of points to cope with the border areas (see above), and the rest of our point budget was "spent" on Phase 2.

Calculating the Number of Points per Interval

If k nondominated points (portfolios) have been computed in the first phase, there are $k - 1$ intervals into which the remaining computation budget has to be placed. Each of these intervals has its own value c_i and m_i. The number of portfolios that is not yet used is $h_2 = t - h_1 - r$. If n_i is the number of points we allocate to interval i, the optimal

distribution – based on the assumption of linearity between the points – is the solution of the following optimization problem:

$$\begin{aligned} \max f(\boldsymbol{n}) &= \sum_{i=1}^{k-1} m_i c_i^2 \frac{n_i}{2(n_i+1)} \\ \text{subject to} & \\ \sum_{i=1}^{k-1} n_i &= h_2 \\ n_i &\in \mathbb{N}_0 \quad \forall i = 1, \ldots, k-1 \end{aligned}$$

This integer optimization problem which at a first glance might look difficult, is fairly easy to solve since the values for $n_1, \ldots, n_{k-1}$ can be computed iteratively: starting with the first point, with the help of Eq. 4.1 we can decide for each point in which interval it is to be placed. The interval with the highest estimated reduction of the approximation value $\Delta F_{\text{ap},j}^{n_j \to n_j+1}$, $j = 1, \ldots, k-1$ receives the point. The only information we need for this is the c_j, m_j and the current n_j for all intervals. If a point is placed in interval j, n_j is increased by one and the value for $\Delta F_{\text{ap},j}^{n_j \to n_j+1}$ has to be updated. Then, without requiring any further computation (besides the search for the highest $\Delta F_{\text{ap},j}^{n_j \to n_j+1}$), the next point can be placed. When the whole remaining budget h_2 has been distributed, the values for E_f are calculated by spreading the portfolios evenly over the intervals. With n_j as the number of portfolios to be placed in Interval j, $E_{j,l}$ as the lower (left) boundary of the interval, and $E_{j,u}$ as the upper (right) boundary, all $E_{f,j}$ in the interval are determined as follows:

$$E_{f,j} = E_{j,l} + \frac{i}{n_j + 1}(E_{j,u} - E_{j,l}) \text{ with } i = 1, \ldots, n_j$$

After calculating the solutions for each of these values of E_f in all intervals, the dominated solutions have to be removed from the union of the three solution sets (i.e. from the results of Phase 1, Phase 2, and the border areas). The remaining (nondominated) solutions form the (hopefully good) approximation of our Pareto front.

Other Distribution Algorithms

The simplest approach, and also the one used most often, is to distribute the available computation budget evenly over the feasible region. In the tests in this chapter we will determine the percentage of the computation budget that can be saved if we replace the naive distribution with the algorithm developed by us.

There are also some completely different approaches how to place the evaluation points to get a good approximation of the complete Pareto front (see e.g. *normal boundary intersection* by Das and Dennis [DD98]). Their main focus, however, is usually put on

problems with more than two objectives and the difficulties resulting from that[4]. For a general overview, the reader is referred to Miettinen [Mie98].

4.3 Mixed-Integer Modelling

Optimization problems are denoted as *mixed-integer problems* if some of the decision variables have to be integral while others are allowed to assume real values. Of special interest for mean-variance portfolio selection are so called quadratic mixed-integer programs (MIQPs). An MIQP is an optimization problem with a quadratic objective function and linear constraints, which may be both equalities and inequalities.

In the last ten years, especially in the years 1996–2000, critical improvements in the field of mixed integer solvers have been made that allow to handle problems which before had been considered unsolvable in a reasonable amount of time. These advances were possible due to the integration of what, up to then, were considered to be mainly theoretical concepts into a production algorithm. For a brief overview on these improvements, the reader is referred to Bixby et al. [BFG+00]. A general overview on integer programming can be found e.g. in Wolsey [Wol98]. Kallrath [Kal02] is a good reference when the reader is interested in the capabilities of mixed-integer modelling techniques.
By adding binary variables, the convex bicriteria problem can be extended to take non-convex constraints into account (see Section 2.5) The additional modifications that are caused by applying the ϵ-Constraint method have no effect on the part of the model that contains the binary variables – and vice versa.

The single-criterion mixed integer problem with a buy-in threshold can therefore be expressed as follows:

ϵ-Constraint mean-variance problem with a buy-in threshold of l_b

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{4.2a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{\mu} \geq E_f \tag{4.2b}$$

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{4.2c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{4.2d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{4.2e}$$

$$l_b \rho_i \leq x_i \leq u \rho_i \tag{4.2f}$$

$$\rho_i \in \{0, 1\} \quad i = 1 \ldots N \tag{4.2g}$$

[4] If there are more than two optimization criteria, one of the main drawbacks of the ϵ-Constraint method becomes more important: In order to get an approximation of a given granularity the number of points that have to be evaluated grows exponentially with the number of optimization criteria. Current publications try to address this problem (see e.g. Laumanns et al. [LTZ06]). As we only have two criteria, this drawback does not affect us.

with binary variables $\rho_i,\ i = 1, \ldots, N$ indicating whether asset i is included in the portfolio or not. (The other variables have the usual meaning. For further details, the reader is referred to Section 2.5.)

Problem instances with a maximum cardinality constraint can be transformed similarly:

ϵ-Constraint mean-variance problem with maximum cardinality K

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{4.3a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{\mu} \geq E_f \tag{4.3b}$$

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{4.3c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{4.3d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{4.3e}$$

$$x_i \leq u_i \rho_i \tag{4.3f}$$

$$\rho_i \in \{0, 1\} \quad i = 1 \ldots N \tag{4.3g}$$

$$\sum_{i=1}^{n} \rho_i \ \leq \ K, \tag{4.3h}$$

where ρ_i is the binary variable that specifies whether asset i should be included in the portfolio, and l_i and u_i are the lower and upper bounds for the weights if the corresponding asset is included.
The two optimization models with buy-in thresholds or maximum cardinality constraints can be input into a solver for mixed integer quadratic programs (MIQP) directly without needing to change anything at all.
The modelling of the 5-10-40-Constraint as MIQP is not as straightforward, however, as a similar ϵ-Constraint transformation still includes a product of variables in Inequality 4.4f:

SMVM with 5-10-40-Constraint

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{4.4a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{\mu} \geq E_f \tag{4.4b}$$

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{4.4c}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{4.4d}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{4.4e}$$

$$\boldsymbol{\rho}^T \boldsymbol{x} \leq 0.4 \tag{4.4f}$$

$$\boldsymbol{x} - 0.05\boldsymbol{\rho} \leq 0.05\boldsymbol{e} \tag{4.4g}$$

$$\rho_i \in \{0, 1\}\ \forall i = 1, \ldots, N \tag{4.4h}$$

$$\boldsymbol{x} \geq \boldsymbol{0} \tag{4.4i}$$

MIQP solvers are unable to cope with constraints of this type and require the introduction of N additional real valued variables:

SMVM with 5-10-40-Constraint (MIQP model)

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} \tag{4.5a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{\mu} = E \tag{4.5b}$$

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{4.5c}$$

$$\boldsymbol{x} - 0.05\boldsymbol{\rho} \leq 0.05\boldsymbol{e} \tag{4.5d}$$

$$\boldsymbol{t} - 0.1\boldsymbol{\rho} \leq \boldsymbol{0} \tag{4.5e}$$

$$\boldsymbol{t} - \boldsymbol{x} \leq \boldsymbol{0} \tag{4.5f}$$

$$\boldsymbol{x} + 0.1\boldsymbol{\rho} - \boldsymbol{t} \leq 0.1\boldsymbol{e} \tag{4.5g}$$

$$\boldsymbol{t}^T \boldsymbol{e} \leq 0.4 \tag{4.5h}$$

$$\boldsymbol{x}, \boldsymbol{t} \geq \boldsymbol{0} \tag{4.5i}$$

$$\rho_i \in \{0, 1\} \qquad i = 1, \ldots, N \tag{4.5j}$$

Thereby, $\rho_i = 1$ implies that asset i is allowed up to 10%, while $\rho_i = 0$ restricts asset i to 5% (Inequality 4.5d). If $\rho_i = 0$, Inequality 4.5e requires $t_i = 0$, otherwise $t_i = x_i$ by Inequalities 4.5e, 4.5f, and 4.5g. Inequalities 4.5d and 4.5h restrict the "heavyweights" to 10%, and their sum to 40%.

Once formulated appropriately, any of the available mixed integer quadratic programming (MIQP) solvers can be used to solve the model. Software packages that contain such a solver can be found at NEOS [NEO06], although – compared to the number of available QP-solvers – there are by far fewer programs with the required capability. In any case, there is no guarantee for getting the optimal portfolio quickly. If there is not enough time to allow the algorithm to terminate regularly, the best feasible portfolio calculated so far can be used instead — a normal practice when working with mixed integer solvers. Of course, this abandons the optimality guarantee.

For further information about the mixed integer approach to the cardinality constrained problem without using an external MIQP-solver, the reader is referred to Bienstock [Bie96], who examined the computational complexity of the problem, tested a self-developed branch-and-cut algorithm using disjunctive cuts, and discussed some of the implementation problems that occurred.

Jobst et al. [JHLM01] use a branch-and-bound algorithm to solve the cardinality constrained problem with buy-in thresholds as well as an index tracking problem, where a portfolio subject to cardinality constraints should perform as closely as possible to an index in terms of expected return and variance. As they calculate many points on the efficient frontier, the available time is too short for solving each MIQP to optimality. Therefore, they limit the number of nodes in the search tree of the branch-and-bound algorithm for each given expected return E_f. To speed up the algorithm further, a previously calculated solution for an adjacent point is used as a warm start solution for the new value of expected return.

Based on our experience, the ability to reuse information gathered in previous solver runs is essential in achieving satisfactory algorithm performance.
Advantages of the mixed integer approaches are certainly that optimality is guaranteed if the algorithm terminates. On the other hand, running time is difficult to predict, and modelling complex constraints is not always straightforward as the example of the 5-10-40-Constraint has demonstrated.
Unfortunately, an MIQP solver can only approximate the Pareto front pointwise; a mixed-integer version of a parametric quadratic programming algorithm is not known. But Chapter 6 describes an algorithm that combines the MIQP methodology with an algorithm that calculates curve-based segments of the Pareto front and merges these segments into a curve-based approximation of the whole front.

4.4 Heuristics Based on the ϵ-Constraint Method

Our main intention in developing the point-based heuristics was to get a feasible solution of fairly good quality in a short time. Feasibility was a core requirement for any solution, as otherwise we could just solve a standard mean-variance problem that does not contain any nonconvex constraints and very quickly get an infeasible solution that is optimal.
Which of the other two design objectives – speed and solution quality – is more important depends on the application context. For this reason, we will report on both criteria in our tests, the quality of the approximation and the time required to calculate it.
Figure 4.6 illustrates the overall procedure of all point-based heuristics in this section.

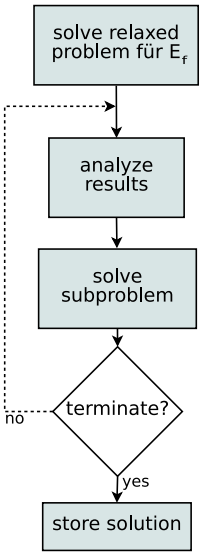


Figure 4.6: Framework for point-based heuristics

The basic assumption behind all of the heuristics in this section is that the feasible region – which is nonconvex – is the union of a collection of convex subsets, each of which is in turn defined by a set of linear constraints. This is true for all three nonconvex problem types we want to solve, and it is the foundation of all of the heuristics that are presented in this section.
We further surmise that the solution of the *relaxed* mean-variance problem, i.e. of the problem that contains all constraints with the exception of the nonconvex one, is a good indication as to which of the convex subsets is the most promising.
With a given lower bound E_f for the expected return, the relaxed problem is passed on to a quadratic programming (QP) solver as a first step. The QP solver – the algorithm that calculates a solution for a quadratic optimization problem with linear constraints – is one of the core components of all ϵ-Constraint heuristics in this chapter. As there are highly optimized codes that provide the required functionality (cf. the NEOS Guide [NEO06]), we have decided – as with the MIQP solver – not to implemented it ourselves. (More information about our choice for the QP solver is provided in Section 4.5.)
The formulation of a model for the relaxed problems is not difficult at all. In case of a problem with the 5-10-40-Constraint and a short sales prohibition, the model looks as follows:

Convex relaxation of a problem with 5-10-40-Constraint:

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} \tag{4.6a}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{\mu} \geq E_f \tag{4.6b}$$

$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{4.6c}$$

$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{4.6d}$$

$$\boldsymbol{x} \leq 0.1\boldsymbol{e} \tag{4.6e}$$

Inequality 4.6e contains the part of the 5-10-40-Constraint that does not compromise the convexity of the feasible region. (The budget constraint can be expressed in terms of Equation 4.6d, any short sales constraint in terms of Inequality 4.6c.)
The relaxations for problems with either cardinality constraint or buy-in thresholds are nearly identical, the only difference is that they lack Ineqequality 4.6e.
In a second step, the solution of the relaxed problem is analyzed and a linearly constrained subset is chosen. The quadratic programming problem with the feasible region defined by the new subset is again passed on to the QP solver. Depending on which heuristic is used, the procedure may terminate after two solver calls with a feasible solution, or the result is analyzed again, and the loop is be repeated till a given termination criterion is fulfilled.
This procedure has to be repeated for many different values of E_f. Therefore the QP solver is called quite frequently. Since only minimal time is required for all the other parts of the algorithm, the solver calls amount to the largest share of the overall algorithm runtime. Hence how often to call the solver is *the* design decision – besides the number of points we use – that directly influences the overall time we need to calculate an approximation.

For problems with a 5-10-40-Constraint, we have developed three heuristics, and both for maximum cardinality constrained problems and problems with buy-in thresholds, two heuristics were designed and tested. In the following, we describe each of them as pseudocode and give some additional explanatory remarks.

4.4.1 Heuristics for Portfolio Selection Problems with 5-10-40-Constraint

The first heuristic for the problems with the 5-10-40-Constraint is called **2-solver-calls heuristics**. It works as documented in Algorithm 1.

Algorithm 1 2-solver-calls heuristic for problems with 5-10-40-Constraint

1: **for all** i **do**
2: $\quad u_i \leftarrow 0.1$
3: **end for**
4: $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$
5: $L_{0.05} \leftarrow \{i\}_{x_i > 0.05}$
6: **if** $\sum_{i \in L_{0.05}} x_i > 0.4$ **then**
7: $\quad$ **for all** $i \notin L_{0.05}$ **do**
8: $\quad\quad u_i \leftarrow 0.05$
9: $\quad$ **end for**
10: $\quad$ Add constraint $r_{0.4} : \sum_{i \in L_{0.05}} x_i \leq 0.4$
11: $\quad \boldsymbol{x} \leftarrow S(\boldsymbol{u}, r_{0.4}, E_f)$
12: **end if**

In the pseudocode of this heuristic, u_i is the upper bound for asset i, E_f, as usual, denotes the lower bound for the expected return. $S(\boldsymbol{u}, E_f)$ and $S(\boldsymbol{u}, r_{0.4}, E_f)$ represent calls of the quadratic programming solver with the parameters $\boldsymbol{u}$ and E_f. Additionally, the second solver call has to pass the constraint $r_{0.4}$ to the solver[5].
One of the main tasks of the heuristic is accomplished in Line 5: the analysis of the solution that has been calculated in the first solver call. There all assets that have been set to a weight larger than 0.05 by the QP solver are added to the set $L_{0.05}$. If the sum of the asset weights in $L_{0.05}$ is smaller than 0.4 (Line 6), the solution of the relaxed problem is feasible for the problem with nonconvex constraints, and the algorithm terminates. Otherwise the upper bound of all assets not in $L_{0.05}$ is set to 0.05 (Lines 7,8) and a restriction $r_{0.4}$ is added, which sets an upper bound of 40% for the sum of all assets in the set $L_{0.05}$. The solution we get from the second solver call is definitely feasible.
As the name indicates, the heuristic requires at most two solver calls for every value of E_f. The main cause for solutions of poor quality is probably the result of the restrictive setting we use for constraint $r_{0.4}$. As a consequence of $r_{0.4}$, it is highly probable that several of the of the assets in $L_{0.05}$ will fall below 5% and wouldn't have to be included

[5]The covariance matrix, the vector of expected returns, the lower bounds for the assets, and all the other linear constraints $\boldsymbol{A_E x} = \boldsymbol{b_e}$, $\boldsymbol{A_I x} \leq \boldsymbol{b_I}$ have to be transferred to the solver as well. We did not include them in our description as they do not change.

in the $r_{0.4}$-Constraint at all. If this happens, the other elements of $L_{0.05}$ are therefore forced to be smaller than they actually would have to be.
In order to cope with this drawback, we have developed the ***n*-solver-calls heuristic**, which is described in Algorithm 2.

Algorithm 2 n-solver-calls heuristic for problems with 5-10-40-Constraint

```
1: for all i do
2:     u_i ← 0.1
3: end for
4: M_l ← ∅, L_{0.05,o} ← ∅, r_0.4 ← ∅
5: finished = false
6: while ¬finished do
7:     x ← S(u, r_0.4, E_f)
8:     L_0.05 ← {i}_{x_i>0.05}
9:     if Σ_{i∈L_0.05} x_i > 0.4 then
10:        M_l ← M_l ∪ {i}_{i∈L_{0.05,o} ∧ i∉L_0.05}
11:        for all i ∈ M_l do
12:            u_i ← 0.05
13:        end for
14:        Modify constraint r_0.4 : Σ_{i∈L_0.05} x_i ≤ 0.4
15:        L_{0.05,o} ← L_0.05
16:    else
17:        finished = true
18:    end if
19: end while
```

In this algorithm, the set $L_{0.05}$ is once more used to store the indices of those assets that have a weight larger than 5%. Additionally, we need the set $L_{0.05,o}$. This set contains all the assets that had a weight larger than 5% in the previous iteration of the *while*-loop (Lines 6–19). The notation $r_{0.4} \leftarrow \emptyset$ describes an operation that sets $r_{0.4}$ as a constraint that is always fulfilled and thus does not influence the solver result.
M_l denotes the set of "locked" assets. Locked assets are assets with an upper bound set to 5%. In the n-solver-calls heuristic, an asset that has been locked can never be unlocked, i.e. the upper bound is never raised. Additionally, we use the termination flag ("*finished*") which indicates feasibility of the solution. When *finished* is set to *true*, the algorithm stops the next time Line 6 is encountered.
The other variables in the algorithm have the same meaning as above.
After initializing M_l, $L_{0.05,o}$, and $r_{0.4}$, the algorithm starts in a similar fashion as the 2-solver-calls heuristic: the same relaxed problem is solved. The solution is again analyzed and $L_{0.05}$ is set to contain all assets with a weight larger than 5%.
If the solution is not feasible with respect to the 5-10-40-Constraint (Line 9), the variables that have had a weight larger than 5% in the previous iteration of the while-loop and have fallen below the 5% barrier in the solution of the current iteration are added to M_l. Then, Constraint $r_{0.4}$ is updated with the new $L_{0.05}$. Finally, $L_{0.05,o}$ is set to the current value of $L_{0.05}$, and the while-loop starts anew.

The number of solver calls in the n-solver-calls heuristic can vary between 1 (if the solution of the relaxed problem is feasible) and a number close to the size of the asset universe (if one asset after another has to be locked). Both extreme cases can occur only theoretically, however. In practice, we have found that the QP solver is called between 3 and 6 times.

A third heuristic for portfolio selection problems with a 5-10-40-Constraint, the **partial derivatives heuristic**, tries to find a middle ground between the previous two. It limits the solver calls to two, but tries to more intelligently select the assets that are limited to 5%. Algorithm 3 describes the procedure.

Algorithm 3 Partial derivatives heuristic for problems with 5-10-40-Constraint

1: **for all** i **do**
2: $\quad u_i \leftarrow 0.1,\ g_i \leftarrow 0$
3: **end for**
4: $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$
5: $L_{0.05} \leftarrow \{i\}_{x_i > 0.05}$
6: $\Sigma_{0.05} \leftarrow \sum_{i \in L_{0.05}} x_i$
7: **if** $\Sigma_{0.05} > 0.4$ **then**
8: $\quad$ **for all** $i \in L_{0.05}$ **do**
9: $\quad\quad g_i \leftarrow \frac{\partial V}{\partial x_i}(0.05 - x_i)$
10: $\quad$ **end for**
11: $\quad$ **for all** $i \notin L_{0.05}$ **do**
12: $\quad\quad u_i \leftarrow 0.05$
13: $\quad$ **end for**
14: $\quad$ **while** $s_{0.05} > 0.4$ **do**
15: $\quad\quad$ Find j with $g_j = \min_{\{i\}} g_i$
16: $\quad\quad L_{0.05} \leftarrow L_{0.05} \setminus j$
17: $\quad\quad \Sigma_{0.05} \leftarrow \Sigma_{0.05} - x_j$
18: $\quad\quad u_j \leftarrow 0.05$
19: $\quad\quad g_j \leftarrow 0$
20: $\quad$ **end while**
21: $\quad$ Add constraint $r_{0.4} : \sum_{i \in L_{0.05}} x_i \leq 0.4$
22: $\quad \boldsymbol{x} \leftarrow S(\boldsymbol{u}, r_{0.4}, E_f)$
23: **end if**

In this algorithm, $\Sigma_{0.05}$ initially denotes the sum of all shares that have a weight larger than 5%. During the algorithm, we subtract from $\Sigma_{0.05}$ the weights of those shares that are newly restricted up to 5% (Line 17). g_i is 0 if asset i has a weight below 5%, or if its upper bound is set to 5%. Otherwise, g_i is set to the estimate of the variance change that occurs if the upper bound of asset i is reduced from 10% to 5% (Line 9). This estimate, $\frac{\partial V}{\partial x_i}(0.05 - x_i)$, is not very precise. It takes into account neither the budget

constraint[6] nor any other linear constraints with the exception of the upper bounds. It can be calculated very quickly, however.
The algorithm uses these estimates to iteratively identify those variables that (hopefully) do the least harm if their upper bound is set to 5%. The upper bounds of the respective variables are then set to 5%, $L_{0.05}$, $\Sigma_{0.05}$, and the respective elements g_i are changed accordingly (Lines 15-19). This is repeated as long as the weights of those assets that remain in $L_{0.05}$ add up to more than 40%. Afterwards, constraint $r_{0.4}$ has to be introduced again to ensure feasibility, as the reduction of several variables enforces an increase in others.

4.4.2 Heuristics for Portfolio Selection Problems with a Maximum Cardinality Constraint

Jobst et al. [JHLM01] describe a simple point-based heuristic for mean-variance portfolio selection problems with a maximum cardinality of k which they called **reoptimization heuristic**. It works as described in Algorithm 4.

Algorithm 4 Reoptimization heuristic for problems with a maximum cardinality of k

1: **for all** i **do**
2: $\quad u_i \leftarrow 1.0$
3: **end for**
4: $L_k \leftarrow \emptyset$
5: $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$
6: Add k assets with k largest x_i to L_k
7: **for all** $i \notin L_k$ **do**
8: $\quad u_i \leftarrow 0$
9: **end for**
10: $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$

After solving the relaxed problem (Line 5)[7], the k assets with the largest portfolio weights in this solution are identified and added to the set L_k. The upper bounds of all shares not in L_k are set to 0 (Line 8). The solution computed in the second solver call is – if the solver is able to calculate one at all – feasible for the portfolio selection problem with maximum cardinality constraint. Exactly two solver calls for every E_f are used by the heuristic. The heuristic is quite similar to the 2-solver-calls heuristic from above.
In the ***n*-solver-calls heuristic** for problems with a maximum cardinality constraint, which we describe in Algorithm 5, we have modified the reoptimization heuristic with the objective to improve solution quality. The price is, once more, a larger number of solver calls.

[6] If the weight of any asset is reduced, another one has to "take up the slack" in order to ensure that the whole budget is spent.

[7] We assumed that there is no stricter upper bound in the problem than 1 and that the lower bound is 0. Any upper bound in the interval $[\frac{1}{k}, 1]$ can be used, however; in that case, Line 2 has to be modified.

Algorithm 5 n-solver-calls heuristic for problems with a maximum cardinality of k

1: **for all** i **do**
2: $\quad u_i \leftarrow 1.0$
3: **end for**
4: $M_l \leftarrow \emptyset$
5: $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$
6: $M_l \leftarrow \{i\}_{x_i<\epsilon}$
7: **for all** $i \in M_l$ **do**
8: $\quad u_i \leftarrow 0$
9: **end for**
10: $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$
11: **while** $|M_l| < N - k$ **do**
12: $\quad$ Find j with $x_j < x_i \forall i \notin M_l$
13: $\quad u_j \leftarrow 0$
14: $\quad M_l \leftarrow M_l \cup \{j\}$
15: $\quad \boldsymbol{x} \leftarrow S(\boldsymbol{u}, E_f)$
16: **end while**

M_l denotes the set that contains the indices of the shares that are forced to become 0, since their upper bounds are set to 0. At the start of the algorithm, M_l is initialized to be empty and the relaxed problem is solved. The obtained solution is analyzed and, in a first step, M_l is set to contain all shares that have a weight smaller than ϵ (Line 6). The value for ϵ depends on the precision of the quadratic programming solver and on the number of investment alternatives. We have found that $10^{-5}\frac{1}{N}$, with N the size of the asset universe, is a reasonable value for ϵ. The intention of this operation is the removal of shares with a weight so small that it has little or no influence on the portfolio variance. If the asset remained in the portfolio, it would occupy one of the k available slots. In the next step, the problem is solved again, but with the newly modified vector of upper bounds. Then, in an iterative procedure, the share with the smallest weight is identified, added to M_l, and its upper bound is set to 0. This is repeated as long as the number of assets not in M_l exceeds the maximum cardinality k.

4.4.3 Heuristics for Portfolio Selection Problems with Buy-In Thresholds

For problems with a buy-in threshold of l_b, we have again designed a simple heuristic that does not use more than 2 solver calls for each E_f. It works as described in Algorithm 6. $\boldsymbol{l}$ denotes the vector of lower bounds for all assets. The solution of the relaxed problem determines if either x_i is set to 0, or its lower bound is raised from 0 to l_b. If x_i is larger than $\frac{l_b}{2}$ in the relaxed solution, then its lower bound is raised (Line 12), otherwise x_i is forced to have zero weight (Line 9). This can be interpreted as a simple "rounding" of the variables.

For the buy-in threshold problem, an improved heuristic – the n-solver-calls heuristic – was developed as well. Algorithm 7 describes its functionality in pseudocode.

Algorithm 6 2-solver-calls heuristic for a problem with buy-in threshold l_b

for all i **do**
    $u_i \leftarrow 1.0$
    $l_i \leftarrow 0$
end for
$L_l \leftarrow \emptyset$
$\boldsymbol{x} \leftarrow S(\boldsymbol{u}, \boldsymbol{l}, E_f)$
$L_l \leftarrow \{i\}_{x_i < \frac{l_b}{2}}$
for all $i \in L_l$ **do**
    $u_i \leftarrow 0$
end for
for all $i \notin L_l$ **do**
    $l_i \leftarrow l_b$
end for
$\boldsymbol{x} \leftarrow S(\boldsymbol{u}, \boldsymbol{l}, E_f)$

Algorithm 7 n-solver-calls heuristic for a problem with buy-in threshold l_b

for all i **do**
    $u_i \leftarrow 1.0$
    $l_i \leftarrow 0$
end for
$\boldsymbol{x} \leftarrow S(\boldsymbol{u}, \boldsymbol{l}, E_f)$
for all i with $x_i < \epsilon$ **do**
    $u_i \leftarrow 0$
end for
for all i with $x_i > l_b$ **do**
    $l_i \leftarrow l_b$
end for
$\boldsymbol{x} \leftarrow S(\boldsymbol{u}, \boldsymbol{l}, E_f)$
while $\exists\, i$ with $0 < x_i < l_b$ **do**
    Find j with $0 < x_j < l_b$ and $x_j \leq x_i\ \forall i$
    Find k with $0 < x_k < l_b$ and $x_k \geq x_i\ \forall i$
    if $x_j < l_b - x_k$ **then**
        $u_j \leftarrow 0$
    else
        $l_k \leftarrow l_b$
    end if
    $\boldsymbol{x} \leftarrow S(\boldsymbol{u}, \boldsymbol{l}, E_f)$
end while

The n-solver-calls heuristic starts by solving the relaxed problem (Line 5)[8]. In the next step, variables with a weight smaller than ϵ are forced to actually become 0 (Lines 6-8). The value for ϵ was – as in the n-solver-calls heuristic for maximum cardinality problems – determined to be $10^{-5}\frac{1}{N}$. Variables with weights exceeding l_b have their lower bounds set to l_b, thus ensuring that they also comply with the buy-in threshold constraint in later solver calls (Lines 9-11).
After getting a new solution based on the more restrictive bounds (Line 12), the one asset that violates the buy-in threshold and is closest to either 0 or the threshold l_b is identified. This variable is from then on set to become 0 (if it was closer to that value) or its lower bound is set to l_b (if the asset was near the threshold). The ensuing problem is solved again, and the loop is repeated as long as there are asset weights that do not comply with the buy-in threshold constraint.

4.5 Test Results for ϵ-Constraint Methods

In this section, we report on the results we achieved with the point-based heuristics and the MIQP solver.
We once more measured the computation time on our benchmark system with an AMD-Athlon-CPU with 1,5 GHz and Linux (Debian sid) as operating system. The program containing all algorithms was compiled with version 3.3 of the Gnu Compiler Collection [GCC06] with the optimization level set to 3. We only report the actual CPU-time that was needed to calculate the solution, as the "real" duration of a run is influenced by the load of the operating system during computation. (Normally the two values do not differ too much as long as the test platform is used exclusively for testing and has sufficient main memory.)
We used CPLEX 9.0 from ILOG [CPL06] to provide both "normal" quadratic programming and MIQP functionality. This should, at least approximately, guarantee the same code maturity level for both utilized solvers. Regarding the conventional QP solver, we examined in a series of preliminary tests how well the barrier optimizer as well as the simplex optimizer from CPLEX (for more details, the reader is referred to the CPLEX user manual [CUs03]) perform. The simplex-based algorithm proved to be significantly faster, most probably due to its superior warm start capabilities. We therefore decided to exclusively use the simplex optimizer in the tests reported in this thesis.
The listed CPU-times are the averages of ten algorithm runs together with the respective standard error. We have to mention that the reported runtimes are strongly influenced by the choice of the QP solver. Therefore, if the CPLEX solver is replaced by another solver, no reliable prediction about the changes of absolute runtimes are possible. The relative order of the computation times should, hopefully, remain the same.
Regarding the solution quality, we again report on both the ideal and the maximum delta-area (cf. Section 4.1).

[8] We again assume that it is possible to invest 100% of the budget in a single asset and that short sales are not allowed. For different upper bounds the algorithm has to be modified accordingly (Line 2).

For all test runs, we used the 2-phase procedure described in Section 4.2 to efficiently distribute our computation budget along the expected return axis. We set our total computation budget for both phases to 400 and used 10% of it in Phase 1.

4.5.1 Problem Instances with 5-10-40-Constraint

Table 4.1 lists the results we obtained for problem instances P1-P7 (see Section 2.4 for more information about the test problems). As in most related publications, we also introduced a short sales prohibition and therefore set the lower bound for all assets to 0. The upper bounds were, due to the 5-10-40-Constraint, set to 10%. Otherwise, besides the budget constraint and the linear constraints that are a consequence of the 5-10-40-Constraint, no further linear equations or inequalities were introduced.

Table 4.1: Test results for the ϵ-Constraint heuristics and the MIQP solver on problem instances with 5-10-40-Constraint

	2-solver-calls	**n-solver-calls**	**part. deriv.**	**MIQP**
P1 ($N = 31$)				
ideal delta-area	6.53e-07	2.95e-07	4.81e-07	2.03e-07
max. delta-area	3.22e-06	2.57e-06	4.11e-06	1.30e-06
CPU-time ± std. error	1.99 ± 0.0033	2.89 ± 0.0050	1.28 ± 0.0023	6.92 ± 0.0052
P2 ($N = 85$)				
ideal delta-area	9.86e-08	6.17e-08	1.76e-07	3.44e-08
max. delta-area	2.21e-06	1.95e-06	4.67e-06	1.19e-06
CPU-time ± std. error	3.98 ± 0.0033	6.16 ± 0.0048	3.13 ± 0.0034	41.75 ± 0.013
P3 ($N = 89$)				
ideal delta-area	6.57e-08	3.52e-08	8.50e-08	1.86e-08
max. delta-area	1.11e-06	8.18e-07	1.63e-06	4.75e-07
CPU-time ± std. error	3.67 ± 0.0028	5.59 ± 0.0045	3.06 ± 0.0051	44.62 ± 0.017
P4 ($N = 98$)				
ideal delta-area	1.56e-07	9.17e-08	1.98e-07	6.44e-08
max. delta-area	2.14e-06	1.71e-06	3.04e-06	1.24e-06
CPU-time ± std. error	3.77 ± 0.0022	5.53 ± 0.0026	3.13 ± 0.0015	69.58 ± 0.383
P5 ($N = 225$)				
ideal delta-area	2.17e-07	9.86e-08	1.44e-07	7.07e-08
max. delta-area	3.34e-06	2.43e-06	2.32e-06	1.63e-06
CPU-time ± std. error	11.48 ± 0.0158	23.33 ± 0.0129	11.59 ± 0.0063	152.4 ± 0.073
P6 ($N = 500$)				
ideal delta-area	1.12e-05	6.23e-06	1.22e-05	4.80e-06
max. delta-area	6.12e-04	4.72e-04	8.07e-04	3.11e-04
CPU-time ± std. error	49.59 ± 0.022	103.2 ± 0.039	56.3 ± 0.028	1622 ± 1.51
P7 ($N = 1000$)				
ideal delta-area	2.00e-05	1.22e-05	2.52e-05	8.43e-06
max. delta-area	2.14e-03	2.06e-03	2.88e-03	1.46e-03
CPU-time ± std. error	175.0 ± 0.27	300.0 ± 0.33	192.6 ± 0.07	8826 ± 3.6

Surprisingly, the mixed-integer solver was able to calculate the 400 points for all problem instances in an acceptable amount of time even though the number of variables had to

be tripled. The problem instance with the largest asset universe (P7) was solved in less than two and a half hours. This amount of time would be acceptable in many application scenarios, especially if we consider that there are much faster computers available nowadays.
The three heuristics (2-solver-calls, n-solver-calls, and partial derivatives heuristic) were able to provide an approximation to the Pareto front in much less time – which was the intent of their design. The 2-solver-calls heuristic is faster than the n-solver-calls heuristic but, as expected, the solution quality is not as good. Disappointingly, when we consider the max. delta-area, the solution quality of the partial derivatives heuristic is even worse than that of the 2-solver-calls heuristic on 6 of the 7 problem instances.

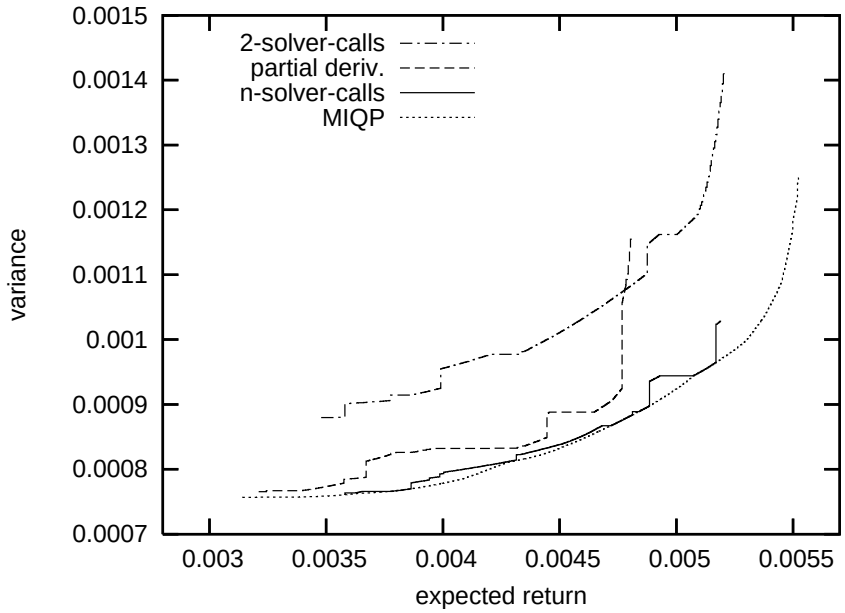
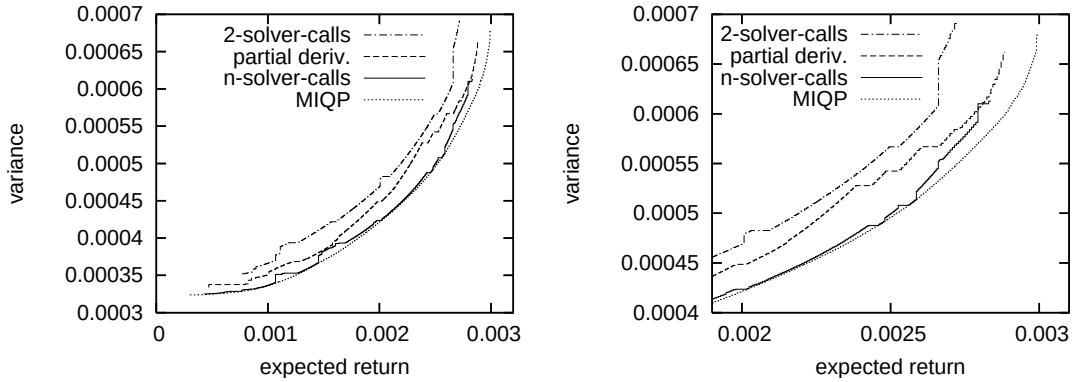


Figure 4.7: Nondominated frontiers calculated by the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for P1 (31 asset universe) with 5-10-40-Constraint

The nondominated frontiers of the four algorithms for Problem 1 (31 assets) are displayed in Figure 4.7. These results demonstrate nicely the main drawback of the partial derivatives heuristic. In the area close to the minimum variance portfolio, the heuristic is better than the 2-solver-calls heuristic. The problem is to be found in the area with high expected returns. There, the partial derivatives heuristic is unable to calculate a feasible solution in a relatively large interval, which in turn results in a poor overall solution quality. This is also the reason why the partial derivatives heuristic is faster than the 2-solver-calls heuristic: infeasibility is usually detected quickly, while the calculation of a feasible solution requires more time. A possible rationale for the poor performance may be found at the core of the partial derivatives heuristic: the assets with small values of $\frac{\partial V}{\partial x_i}(0.05 - x_i)$ get an upper bound of 5%. This is, however, only appropriate if variance

reduction is the main concern. Should the maximization of the expected return be more important, the technique will quite often choose the wrong assets for a reduction.
Similar results can be observed for the other problem instances[9] – with one exception: for P5 (225 assets), the partial derivatives heuristic surpasses even the n-solver-calls heuristic, at least when we only look at the max. delta-area. Figure 4.8(a) shows that the n-solver-calls heuristic is superior on nearly the whole breadth of the expected return axis. Therefore, the ideal delta-area is smaller for the n-solver-calls heuristic. The explanation why the results are different for the max. delta-area is to be found at the right edge (Figure 4.8(b)), where the partial derivatives heuristic is able to return feasible solutions beyond the end of the n-solver-calls heuristic. The values at both ends of the curves, and especially at the right one, have a huge impact on the max. delta-area (cf. Section 4.1). In this case, the effect was large enough to offset the otherwise inferior performance along the rest of the Pareto front.



(a) Complete fronts

(b) Right edge of the fronts

Figure 4.8: Nondominated frontiers calculated by the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for P5 (225 asset universe) with 5-10-40-Constraint

This problem instance demonstrates that blindly trusting only one of the highly aggregated results provided by any of the two area measures can be misleading. We therefore recommend to graphically compare the results of the heuristics in ambiguous cases.
Appendix A.1 contains corresponding figures for the other problem instances (P2, P3, P4, P6, and P7).

4.5.2 Problem Instances with Maximum Cardinality Constraint

Table 4.2 lists the results for problem instances P1-P7 (see Section 2.4 for more details) when a maximum cardinality constraint was introduced. Again, short sales were not allowed. Apart from the ever present budget constraint, no further linear equations or

[9]The difference is, however, not always as pronounced as in the first problem instance.

inequalities were introduced. We limited the maximum number of different shares the portfolio may contain to 4 for the smaller problem instances P1–P4, and to 8 for the larger problems P5–P7.
Both the reoptimization heuristic and the n-solver-calls heuristic are able to rapidly compute an approximation of the Pareto front of fairly good quality. The n-solver-calls heuristic again yields better results but nearly quadruples computation time. Problem 3 (89 asset universe; Figure 4.9) is one instance in which results of the reoptimization heuristic are clearly inferior, while for P7 (1000 asset universe; Figure 4.10), it is difficult to even visually discern the different curves. All front approximations are much closer to the ideal front than in problems with a 5-10-40-Constraint: the max. delta-area is by far smaller even though the Pareto fronts for cardinality constrained problems stretches across a larger distance in the mean-variance space[10].
The figures depicting the nondominated fronts for the P1, P3, P4, P5, and P6 are again provided in Appendix A.2. The results for these problem instances vary slightly, but P3 is the one instance in which the reoptimization heuristic performs worst.
Summarizing: the nondominated fronts calculated by the n-solver-calls heuristic are always very close to the ideal front, while there are a few areas where the reoptimization heuristic is unable to calculate efficient solutions (cf. Figure 4.9 for $E_f \in [0.0058, 0.0063]$). Nevertheless, the results for both heuristics are surprisingly good.

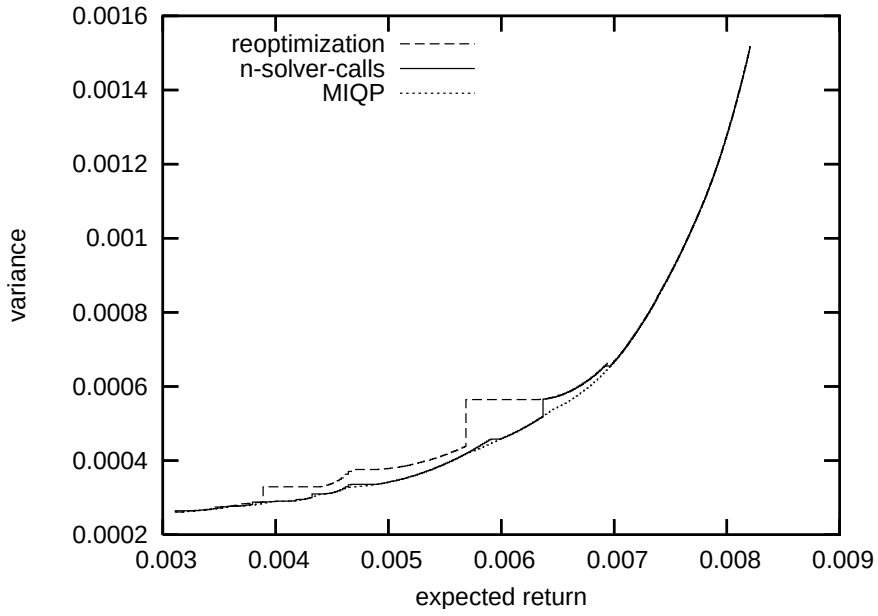


Figure 4.9: Nondominated frontiers calculated by the reoptimization heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 3 (89 assets) with a maximum cardinality of $K = 4$

[10]The longer Pareto fronts are the result of the nonexisting upper bounds for problems with a cardinality constraint.

Table 4.2: Test results for the ϵ-Constraint heuristics and the MIQP solver on problems with cardinality constraints. Maximum cardinality was set to $K = 4$ for problems P1–P4, and to $K = 8$ for problems P5–P7.

	reoptimization	**n-solver-calls**	**MIQP**
P1 ($N = 31$)			
ideal delta-area	1.93e-07	1.92e-07	1.67e-07
max. delta-area	2.85e-07	2.83e-07	2.58e-07
CPU-time ± std. error	0.31 ± 0.0026	0.453 ± 0.0015	3.93 ± 0.0033
P2 ($N = 85$)			
ideal delta-area	6.67e-07	5.12e-07	4.55e-07
max. delta-area	1.05e-06	8.91e-07	8.24e-07
CPU-time ± std. error	1.465 ± 0.0017	4.496 ± 0.0062	764.6 ± 0.12
P3 ($N = 89$)			
ideal delta-area	3.65e-07	2.40e-07	2.19e-07
max. delta-area	5.83e-07	4.58e-07	4.38e-07
CPU-time ± std. error	1.835 ± 0.0022	5.368 ± 0.0039	2079 ± 0.33
P4 ($N = 98$)			
ideal delta-area	7.82e-07	5.90e-07	5.36e-07
max. delta-area	1.09e-06	8.80e-07	8.19e-07
CPU-time ± std. error	2.117 ± 0.0021	7.707 ± 0.0062	78474 ± n.a.[1]
P5 ($N = 225$)			
ideal delta-area	2.10e-08	1.61e-08	1.42e-08
max. delta-area	4.71e-08	3.21e-08	3.01e-08
CPU-time ± std. error	5.72 ± 0.0033	8.077 ± 0.0047	194.6 ± 0.094
P6 ($N = 500$)			
ideal delta-area	2.058e-06	1.31e-06	n.a.
max. delta-area	3.95e-06	2.32e-06	n.a.
CPU-time ± std. error	28.76 ± 0.0074	47.86 ± 0.014	n.a.
P7 ($N = 1000$)			
ideal delta-area	6.98e-06	5.58e-06	n.a.
max. delta-area	2.19e-05	1.56e-05	n.a.
CPU-time ± std. error	109.9 ± 0.19	204.5 ± 0.44	n.a.

[1] Due to the extremely long algorithm runtime, the configuration was started only once. Therefore the standard error could not be calculated.

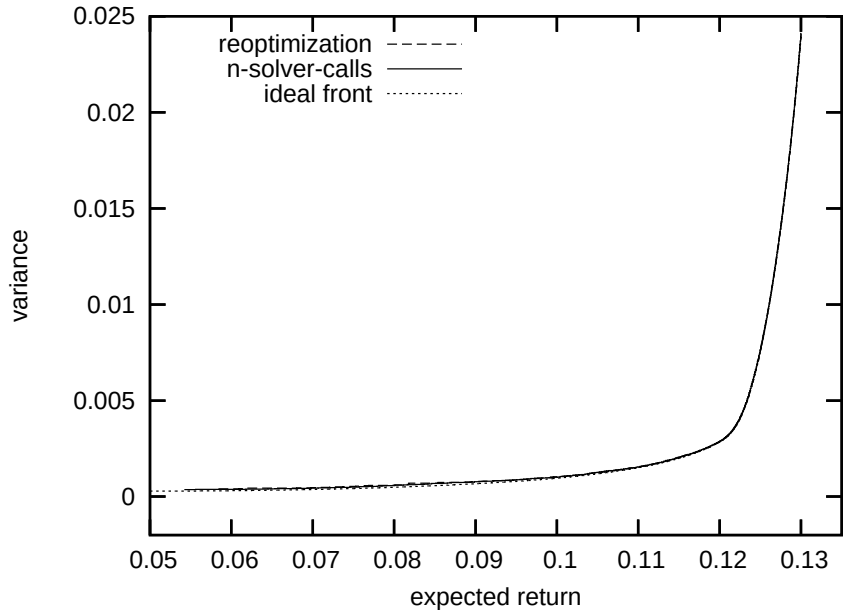


Figure 4.10: The ideal front and the results of the reoptimization heuristic and the n-solver-calls heuristic for Problem 7 (1000 assets) with maximum cardinality of $K = 8$.

Problems with cardinality constraints seem to pose a far stronger challenge for the CPLEX MIQP solver than those with a 5-10-40-Constraint, since the computation time is far longer. For the two largest problem instances, we were unable to get results in a reasonable amount of time. And, when we look at the results from the two heuristics, there doesn't seem to be much potential for improvement anyway: e.g. in Figure 4.10, we printed the solution of the ideal front instead of the MIQP solution we did not have, and the difference between two heuristics and the ideal front is hard to discern. Only when sections of the front are scaled up the discrepancy gets more visible.
One drawback of the MIQP solver is demonstrated by the results of P4: it is not always obvious which problems require longer computation times. The solution of P4 with a maximum cardinality of 4 took nearly a day to run through, while the solution of Problem 5 with maximum cardinality of 8 could be computed in little more than 3 minutes. The runtimes of the heuristics are much more consistent, with a slight edge for the reoptimization heuristic, since the number solver calls is predetermined.

4.5.3 Problem Instances with Buy-In Thresholds

Table 4.3 lists the results that were obtained for the problem instances P1-P7 (cf. 2.4) with an additional buy-in threshold of 0.05. Again, short sales were not allowed ($x_i \geq 0 \; \forall i$), and we also did not use any upper bounds. As always, the budget constraint was introduced to normalize the results.

Table 4.3: Test results for the ϵ-Constraint heuristics and MIQP solver for problems P1–P7 with a buy-in threshold of 0.05.

	2-solver-calls	**n-solver-calls**	**MIQP**
P1 ($N = 31$)			
ideal delta-area	7.18e-08	7.14e-08	6.75e-08
max. delta-area	7.21e-08	7.17e-08	6.78e-08
CPU-time ± std. error	0.391 ± 0.0018	0.478 ± 0.002	5.893 ± 0.0037
P2 ($N = 85$)			
ideal delta-area	3.71e-08	3.52e-08	3.35e-08
max. delta-area	5.54e-08	5.33e-08	5.18e-08
CPU-time ± std. error	1.551 ± 0.0023	3.562 ± 0.0042	54.06 ± 0.014
P3 ($N = 89$)			
ideal delta-area	3.03e-08	2.51e-08	2.36e-08
max. delta-area	3.92e-08	3.40e-08	3.24e-08
CPU-time ± std. error	1.942 ± 0.002	4.813 ± 0.003	120.6 ± 0.021
P4 ($N = 98$)			
ideal delta-area	4.92e-08	3.90e-08	3.73e-08
max. delta-area	6.50e-08	5.02e-08	4.87e-08
CPU-time ± std. error	2.337 ± 0.003	6.168 ± 0.012	238.8 ± 0.043
P5 ($N = 225$)			
ideal delta-area	6.78e-09	6.38e-09	5.47e-09
max. delta-area	9.64e-09	9.24e-09	8.33e-09
CPU-time ± std. error	6.63 ± 0.0037	9.112 ± 0.0049	291.6 ± 0.11
P6 ($N = 500$)			
ideal delta-area	9.58e-07	6.66e-07	5.28e-07
max. delta-area	1.21e-06	9.20e-07	7.49e-07
CPU-time ± std. error	41.58 ± 0.010	54.07 ± 0.023	3611 ± 0.63
P7 ($N = 1000$)			
ideal delta-area	3.94e-06	3.21e-06	2.96e-06
max. delta-area	7.27e-06	4.96e-06	4.72e-06
CPU-time ± std. error	156.8 ± 0.27	219.6 ± 0.49	22919 ± n.a.[1]

[1] Due to the extensive algorithm runtime, this configuration was solved only once. As a consequence, the standard error could not be calculated.

Both heuristics, the 2-solver-calls heuristic and the n-solver-calls heuristic, did extremely well. Their results are – apart from tiny deviations – so close to the solution of the MIQP solver to be visually indistinguishable on all problem instances (see e.g. Figure 4.11). The diagrams for the other problem instances are provided in the appendix in Section A.3.
The max. delta-areas are even smaller than those calculated for portfolio selection problems with cardinality constraints. A significant part of it is made up of the ideal delta-area, i.e. the influence of the values at the edges of the Pareto front is not as pronounced as is was for problems with 5-10-40-Constraint.
The results of the n-solver-calls heuristic surpassed those of the 2-solver-calls heuristic on all test problems, but again at the cost of longer computation times.

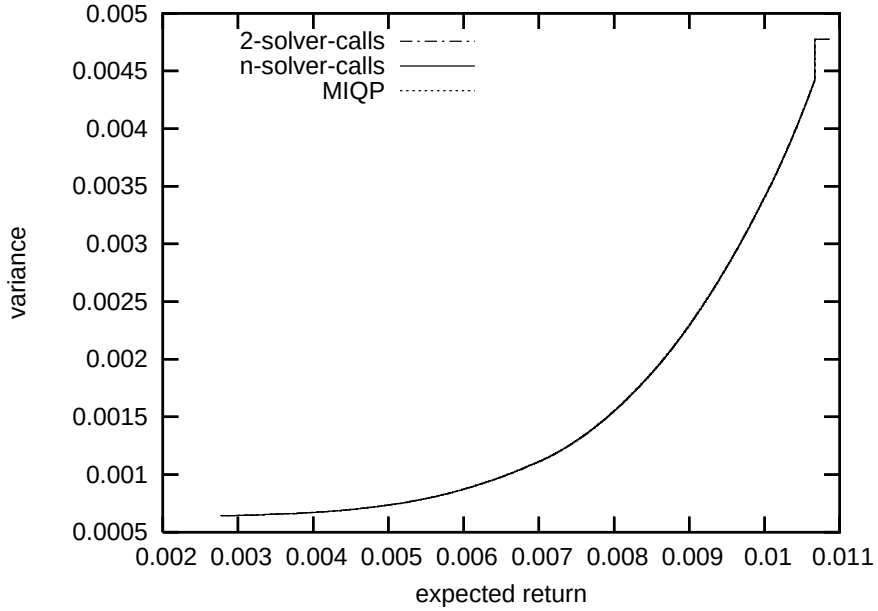


Figure 4.11: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 1 (31 assets) with a buy-in threshold of 0.05

The MIQP solver was able to provide results for all problem instances as well, but the computation time for the larger problems was quite extensive. Considering that the heuristics are able to compute results that are quite close in a small fraction of the time, the only benefit of the MIQP solver seems to be that it guarantees optimality.
One curious effect that can be observed to some degree on all problem instances with buy-in thresholds is a "jump" at the right edge of the Pareto front. This jump is easily explained: the rightmost point of the Pareto front is the one for which 100% of the budget is invested in the asset with the highest expected return. As each share that is included in the portfolio has to reach at least the threshold l_b, no feasible solutions can exist between this rightmost point and the point that has a share of $1 - l_b$ invested in the asset with the highest expected return and a share of l_b invested in the one with the second highest expected return.

4.5.4 Performance Improvement by the 2-Phase Procedure

In all previous tests in this chapter, we used the 2-Phase procedure from Section 4.2 to determine a series of values for E_f. In this section, we report on tests that try to estimate how many additional points are necessary to achieve a solution of equivalent quality when we just distribute the points evenly in the interval $[E_{\min}, E_{\max}]$.
In the results reported above, a budget of 400 points had been used in all tests, with 40 points in the first phase, and 360 in the second. For the following set of tests we have varied the point budgets in the range between 500 to 1200, with a stepsize of 50 (i.e. the first test had 500 points, the second 550 points, ...). The points have been distributed equidistantly between $E_{\min}$ and $E_{\max}$, which we have determined as described in Section 4.2.
We have also calculated a linear approximation for the set of pairs $(n, F(n))$ consisting of the number of points and the associated solution quality. Figure 4.12 illustrates this for Problem 3 and the 2-solver-calls heuristic.

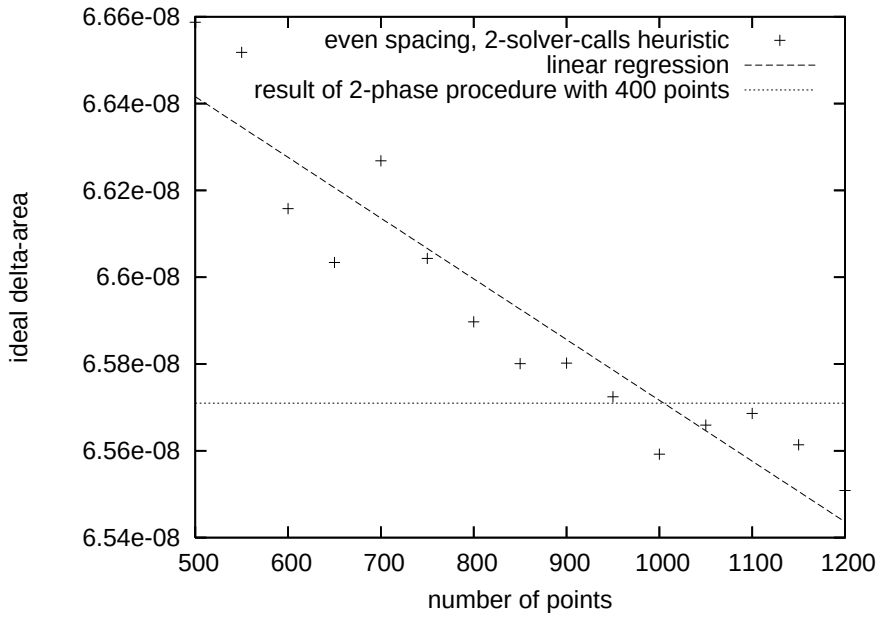


Figure 4.12: Ideal delta-area for 2-solver-calls heuristic with varying point budgets and equidistantly spaced values of E_f. Linear regression for the points, and the result of the 2-Phase procedure for 400 points.
Problem instance: P3 (89 assets) with 5-10-40-Constraint

With this approximation, we have interpolated the number of points we would have needed to achieve the same solution quality as the 2-Phase procedure with 400 points. With this interpolation, we have then calculated the multiples based on the original 400 point count, e.g. $945/400 \approx 2.36$. In order to achieve comparable results, the upper and

lower bounds as well as maximum cardinality and buy-in thresholds have been set to the same values we had used in the previous tests in Chapter 4.
For the problems with 5-10-40-Constraint, Table 4.4 reports on the calculated multiples and on the coefficients of determination (i.e. the squared correlation coefficient) R^2 that measure regression quality. Values with $R^2 < 0.5$ are not reported. As the max. delta-area is strongly influenced by the rightmost portfolio, a slight horizontal shifting of this portfolio can distort the result in a massive way. This caused problems for the 3 heuristics developed for the problem type with 5-10-40-Constraint: no reliable linear trend could be identified for the max. delta-area, as the coefficient of determination was consistently below 0.4. We therefore only print the multiples based on the ideal delta-area.
In case of the MIQP solver, the right edge of the front has been determined in advance by solving a mixed-integer linear program. Therefore, the expected return of the rightmost portfolio is always the same, irrespective of the computation budget. As a consequence, with the MIQP solver, the results for the max. delta-area are nearly identical to the ones from the ideal delta-area. They are reported in Table 4.4 as well.

Table 4.4: Multiples required to achieve similar solution quality with equidistant spacing for problems with 5-10-40-Constraint. (Basing point: results obtained with 2-Phase procedure and 400-point budget)

	2-solver-calls	**n-solver-calls**	**part. deriv.**	**MIQP**
P1 ($N = 31$)				
ideal (max.) delta-area	2.93	n.a.[1]	2.71	1.53 (1.54)
R^2: ideal (max.)	0.87		0.82	0.92 (0.93)
P2 ($N = 85$)				
ideal delta-area	3.12	2.74	2.87	1.46 (1.43)
R^2: ideal (max.)	0.94	0.52	0.55	0.95 (0.95)
P3 ($N = 89$)				
ideal delta-area	2.51	2.95	2.93	1.38 (1.38)
R^2: ideal (max.)	0.87	0.76	0.52	0.95 (0.95)
P4 ($N = 98$)				
ideal delta-area	2.46	2.1	2.48	1.28 (1.28)
R^2: ideal (max.)	0.93	0.66	0.59	0.94 (0.94)
P5 ($N = 225$)				
ideal delta-area	3.2	3.14	3.25	1.05 (1.08)
R^2: ideal (max.)	0.92	0.92	0.84	0.96 (0.96)
P6 ($N = 500$)				
ideal delta-area	2.7	1.8	3.03	2.3 (2.31)
R^2: ideal (max.)	0.79	0.52	0.56	0.94 (0.94)
P7 ($N = 1000$)				
ideal delta-area	2.7	2.52	2.93	1.93 (1.93)
R^2: ideal (max.)	0.92	0.51	0.72	0.94 (0.92)

[1] The value of R^2 was well below 0.5 in this case. Therefore no reliable interpolation could be calculated.

In Table 4.5, we document the results for similar tests that were performed on problems with a maximum cardinality constraint. As was indicated in Fig. 4.9 and Fig. 4.10,

the right edges of the heuristics and the MIQP solver are – for all practical purposes – situated on the ideal front, so that the results (coeff. of determination and multiples) of the ideal delta-area and the max. delta-area do not differ much. Both are reported in Table 4.5. Due to extensive runtimes of the MIQP solver variant, the tests for Problem 4 have not been carried out, as they alone would have required nearly one month computation time.

Table 4.5: Multiples required to achieve the same solution quality with equidistant spacing for problems with a maximum cardinality constraint. (Basing point: results obtained with 2-Phase procedure and 400-point budget)

	reoptimization	**n-solver-calls**	**MIQP**
P1 ($N = 31$)			
ideal (max.) delta-area	1.78 (1.75)	1.76 (1.73)	1.41 (1.39)
R^2: ideal (max.)	0.94 (0.94)	0.94 (0.94)	0.94 (0.94)
P2 ($N = 85$)			
ideal (max.) delta-area	3.18 (3.15)	2.84 (2.82)	2.36 (2.34)
R^2: ideal (max.)	0.93 (0.93)	0.93 (0.93)	0.94 (0.94)
P3 ($N = 89$)			
ideal (max.) delta-area	2.34 (2.34)	2.28 (2.28)	1.22 (1.22)
R^2: ideal (max.)	0.92 (0.92)	0.96 (0.96)	0.94 (0.94)
P4 ($N = 98$)			
ideal (max.) delta-area	2.66 (2.6)	2.54 (2.54)	n.a.
R^2: ideal (max.)	0.75 (0.75)	0.94 (0.94)	
P5 ($N = 225$)			
ideal (max.) delta-area	3.37 (3.36)	3.36 (3.36)	2.13 (2.13)
R^2: ideal (max.)	0.94 (0.94)	0.94 (0.94)	0.94 (0.94)
P6 ($N = 500$)			
ideal (max.) delta-area	2.69 (2.59)	3.02 (2.99)	n.a.
R^2: ideal (max.)	0.94 (0.94)	0.94 (0.94)	
P7 ($N = 1000$)			
ideal (max.) delta-area	3.49 (2.96)	3.51 (3.51)	n.a.
R^2: ideal (max.)	0.94 (0.94)	0.94 (0.94)	

Table 4.6 reports the multiples for problems with a buy-in threshold of 0.05. For this problem type, the calculated multiples and values for R^2 show no significant differences regardles whether they are based on the max. delta-area or the ideal delta-area, at least when only the numerical precision displayed in Table 4.6 is considered. Thus it is sufficient to report only a single value.

For the heuristics (2-solver-calls, n-solver-calls, reoptimization, or partial derivatives) the calculated multiples for all three problem types are nearly always above 2.0, i.e. the 2-Phase allows to save at least 50% of the computational "expenditure" in comparison to the naïve approach of equidistant spacing. The only exception is the smallest instance (P1) for both the maximum cardinality constrained problem and the problem with buy-in thresholds. But even in these two cases, the multiples were still 1.7 or higher.

Table 4.6: Multiples required to achieve the same solution quality with equidistant spacing for problems with a buy-in threshold of 0.05. (Basing point: results obtained with 2-Phase procedure and 400-point budget)

	2-solver-calls	**n-solver-calls**	**MIQP**
P1 ($N = 31$)			
ideal/max. delta-area	1.7	1.7	1.28
R^2: ideal/max.	0.93	0.93	0.93
P2 ($N = 85$)			
ideal/max. delta-area	3.03	3.03	2.29
R^2: ideal/max.	0.9	0.9	0.9
P3 ($N = 89$)			
ideal/max. delta-area	2.28	2.34	1.35
R^2: ideal/max.	0.95	0.94	0.94
P4 ($N = 98$)			
ideal/max. delta-area	2.59	2.61	1.9
R^2: ideal/max.	0.92	0.92	0.93
P5 ($N = 225$)			
ideal/max. delta-area	3.44	3.44	2.15
R^2: ideal/max.	0.89	0.89	0.93
P6 ($N = 500$)			
ideal/max. delta-area	2.85	2.99	2.46
R^2: ideal/max.	0.89	0.89	0.94
P7 ($N = 1000$)			
ideal/max. delta-area	3.46	3.45	n.a.
R^2: ideal/max.	0.88	0.88	

The multiples calculated for the MIQP solver are not as high, but they are still well above 1. The reason why the MIQP approach does not benefit as much is probably to be found in the different determination of the boundaries: if no MIQP solver is available, it is not possible to determine the nondominated portfolio with the lowest variance without using up part of the point budget. In case of evenly spaced points, a large amount of the budget is wasted in the interval between $E_{\min}$ and expected return of the minimum variance portfolio. With the 2-Phase procedure, the same happens in the first phase, but since we only use a small part of the budget for this, not many points are wasted. In Phase 2, nothing of the budget is squandered beyond the minimum variance portfolio as the points are distributed according to the expected gain – which is 0 between $E_{\min}$ and the expected return of the minimum variance portfolio.
The heuristics therefore benefit in two ways from the 2-Phase procedure: by (a) a more efficient determination of the minimum variance portfolio and (b) the effective distribution of the bulk of the budget. The MIQP solver can only profit from the second advantage, since the first functionality is already provided by the MIQP itself, irrespective if the 2-Phase distribution is used or not.

Nevertheless, our overall conclusion is that regardless if we use heuristics or an MIQP solver to calculate a single point, the 2-Phase procedure is able to produce equivalent results in significantly less time.

4.6 Summary: ϵ-Constraint Heuristics

In this chapter we have described heuristics that are able to calculate a point-based approximation of the Pareto front for portfolio selection problems with nonconvex constraints. These heuristics have all been based on the ϵ-Constraint method which changes the bicriteria mean-variance problem into a single-criterion problem by setting a lower bound for one of the criteria. In a mean-variance setting, the expected return is usually the criterion that is transformed into an additional constraint, and the minimization of the variance remains the sole objective.
The two main tasks,

1. the distribution of the points along the expected return axis
2. the minimization of the variance with respect to both convex and nonconvex constraints

have been tackled separately. We have specified an algorithm that efficiently allocates a given budget of search points along the expected return axis with the goal to reduce the approximation error.
We have also presented several relatively simple but nevertheless effective heuristics that quickly calculate a feasible solution of acceptable quality for the single-criterion mean-variance problem with nonconvex constraints. Additionally, we have shown how a problem with 5-10-40-Constraint can be modelled in a way that allows quadratic mixed integer solvers to be used in place of the heuristics[11].
Test results indicate that the heuristics are quite effective for problems with maximum cardinality constraint or buy-in thresholds. The results for those two problem types were very close to the theoretical optimum, with the more time-consuming heuristics usually yielding slightly better results. The quality of the heuristic results for problems with 5-10-40-Constraint was, compared to the other two problem types, not as good, but again the more time-consuming heuristics were superior with respect to solution quality.
We have also reported the results for all three problem types when the problem-specific heuristics were replaced by an MIQP solver. Surprisingly, the MIQP solver was quite fast for problems with 5-10-40-Constraint, while several problem instances with buy-in thresholds or especially with a maximum cardinality constraint needed much more time.
Furthermore, we have tried to estimate how much of the computational budget can be saved when the 2-Phase algorithm for point distribution is used instead of naïvely placing the points in evenly spaced intervals. The savings were generally higher for the heuristics than for the MIQP solver, as in case of the heuristics, the algorithm avoids wasting large

[11] For problems with buy-in thresholds or cardinality constraints, no significant changes to the models presented in Section 2.5 are necessary for them to be processed by an MIQP solver.

sections of the budget on areas that don't contain any nondominated solutions. When an MIQP solver is available, the exact right and left end of the Pareto front can be calculated easily, with the consequence that no part of the budget is wasted beyond the edges of the Pareto front even if we use the simple distribution scheme.

5 An Envelope-Based Multi-Objective Evolutionary Algorithm

This chapter is focused on generating a solution for portfolio selection problems with nonconvex constraints as well, but the approach we propose is quite different to those discussed in Chapter 4. The algorithms described in the previous chapter have one commonality: they are all based on the ϵ-Constraint method, i.e. they set a lower bound for the expected return and thus turn this criterion into an additional linear inequality. Since the solution of such a model only represents a single point on the efficient frontier, the single-point algorithms have to be applied many times with different expected returns in order to obtain a reasonable approximation of the true efficient frontier.
Some heuristic approaches like evolutionary algorithms allow to consider multiple objectives simultaneously, generating a set of solutions approximating the efficient frontier in one run. However, they are still point-based, generating a finite set of alternative portfolios.
What we suggest in this chapter is to combine a multi-objective evolutionary algorithm (MOEA) with an embedded parametric quadratic programming (PQP) algorithm in order to solve these portfolio selection problems with nonconvex constraints. The idea is to let the evolutionary algorithms handle the nonconvex constraints and to basically divide the problem into a set of problems with convex constraints, which can be solved optimally with a PQP algorithm. The overall solution to the problem is then generated by combining the solutions of the individual convex problems. As we will demonstrate, the approach significantly outperforms other state-of-the-art evolutionary algorithms. Furthermore, to our knowledge, this is the first approach to nonconvex portfolio selection yielding a continuous solution set, as opposed to the discrete solution set generated by the point based approaches.
The basic problem type considered in this chapter remains the one formalized as the Standard Mean-Variance Model (SMVM):

SMVM

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} \tag{5.1a}$$
$$\max E(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\mu} \tag{5.1b}$$
subject to
$$\boldsymbol{A_I x} \leq \boldsymbol{b_I} \tag{5.1c}$$
$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{5.1d}$$
$$\boldsymbol{A_E x} = \boldsymbol{b_E} \tag{5.1e}$$

with one of two types of additional (nonconvex) constraints added to the problem: either the 5-10-40-Constraint or a maximum cardinality constraint. (For more details on the model and the different constraints, the reader is referred to Chapter 2.) We did not consider buy-in thresholds in this chapter, although constraints of that kind could be handled by our new approach in a straightforward manner[1].
In order to make the results comparable to those already calculated, we replaced the general inequalities by upper bounds of 1 and lower bounds of 0 for all individual shares:

$$\min V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} \tag{5.2a}$$

$$\max E(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\mu} \tag{5.2b}$$

subject to

$$\boldsymbol{x}^T \boldsymbol{e} = 1 \tag{5.2c}$$

$$0 \leq x_i \leq 1 \qquad i = 1, \ldots, N \tag{5.2d}$$

(Needless to say, the nonconvex constraints have still to be added to the problem formulation.)
The remainder of this chapter is in large parts based on Branke et al. [BSS+06]. It is structured as follows: In Section 5.1, we provide a brief overview on related approaches that use metaheuristics to solve portfolio selection problems that are subject to nonconvex constraints. A short introduction to multi-objective evolutionary algorithms is provided in Section 5.2, followed in Section 5.3 by a short description of a state-of-the-art point-based MOEA which will be used as a reference later on. Our approach, called envelope-based multi-objective evolutionary algorithm (E-MOEA), is presented in Section 5.4. Section 5.5 reports on the empirical evaluation of our approach. This chapter concludes in Section 5.6 with a brief summary.

5.1 Metaheuristics: Related Approaches

Because of the potentially extensive running times of the mixed integer approaches (for runtime tests, see Section 4.5), and due to either their complex implementation (when implemented by oneself), or due to the high prices of commercial mixed integer packages, many researchers have turned to metaheuristics. The term *metaheuristic* usually describes a generic optimization principle that is widely applicable to many different problem domains. Quite often, the basic idea behind a metaheuristic is inspired by a natural phenomenon, be it a physical process like simulated annealing (SA) or biologically inspired algorithms like ant colony optimization (ACO) or evolutionary algorithms (EAs). Like other heuristics, such metaheuristics do not guarantee optimality, but it is hoped that they quickly find solutions of reasonable quality.
There are several publications discussing the use of metaheuristics to solve nonconvex portfolio selection problems. Most of them apply the metaheuristic just to solve the

[1] The results from Chapter 4 indicate that buy-in thresholds don't seem to modify the Pareto front significantly when they are introduced. The approximation is nearly identical to the ideal front.

MIQP with a given desired expected return, i.e. to generate a single point of the efficient frontier in mean-variance objective space.
Chang et al. [CMBS00] compare an EA, tabu search, and SA to solve portfolio selection problems in which each solution has to contain a predetermined number of assets. Because none of the approaches turned out to be a clear winner, they suggest to run all three and combine the results. Schaerf [Sch02] improves on the work of Chang et al. by testing several neighborhood relations for tabu search. Crama and Schyns [CS03] apply simulated annealing to a portfolio selection problem with cardinality constraints as well as turnover and trading restrictions. They particularly focused on ways to handle constraints, partially by enforcing feasibility, partially by introducing penalties. A hybrid between SA and EA is proposed in Maringer and Kellerer [MK03]. In Maringer [Mar02b], ACO is used only to determine the relevant assets for a cardinality-constrained portfolio selection problem, while the weights are determined with a conventional QP solver. The only authors who, to our knowledge, address the 5-10-40-Constraint are Derigs and Nickel [DN01]. They use SA for optimization, but do not provide details on how they ensure feasibility of solutions. In a later publication [DN03], they expand their previous work, but the focus is put on developing a decision support system for portfolio selection. In [EKS04], a problem with five objectives and nonconvex constraints is considered, but the five objectives are accumulated into one based on multiattribute utility theory. Then local search, SA, tabu search and an EA are used to solve the resulting single-objective mixed-integer problem.
Eddelbüttel [Edd96] uses a QP solver within a genetic algorithm to solve the index-tracking problem. Here, the GA determines which assets should be included in the portfolio, while the optimal weights are calculated by the QP solver.
Instead of setting the expected return E as constant E_f and then solving a separate single-objective optimization problem for several E_f as do all of the above methods and as was done in Chapter 4 as well, Streichert et al. [SUZ03, SUZ04a, SUZ04b], apply a multi-objective evolutionary algorithm (MOEA). MOEAs are variants of EAs that exploit the fact that EAs work with a population of solutions, and simultaneously search for a set of efficient alternatives in a multi-objective setting. (See also the Section 5.2.) Note that while this approach generates several solutions along the efficient frontier in one run, it is still point-based, i.e. it only generates a discrete set of solutions. Since we will use this algorithm for empirical comparison with our approach, it will be discussed in more depth in Section 5.3.
Other publications reporting on the use of MOEAs for portfolio selection include Lin and Wang [LW02] who consider roundlots (cf. Section 2.5, and Fieldsend et al. [FMP04], who address the cardinality issue but consider the number of allowed assets, K, as a third objective to be minimized. Armananzas and Lozano [AL05] apply multi-objective variants of greedy search, SA and ACO to the cardinality-constrained portfolio selection problem. Schlottmann and Seese [SS05] consider credit portfolio optimization and use a gradient-based local search within their MOEA.
A recent survey on metaheuristics in financial applications can be found in Schlottmann and Seese [SS04].

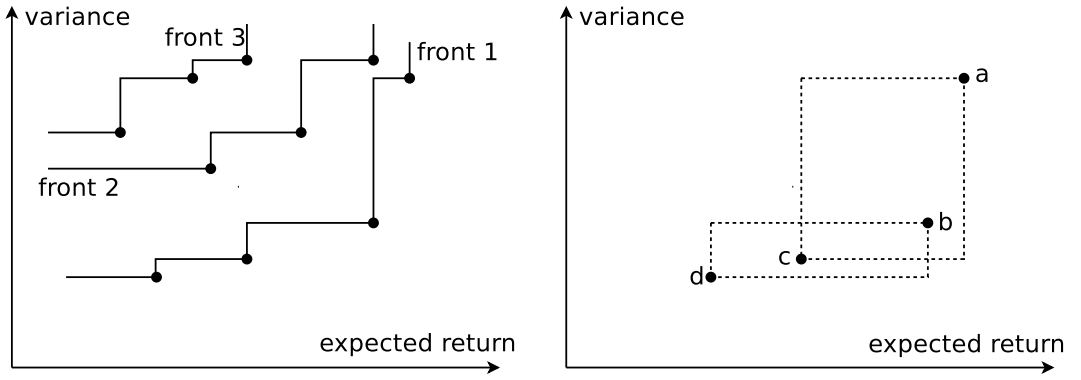
A general advantage of metaheuristic approaches is certainly their flexibility. It would be straightforward e.g. to use an alternative risk measure. Also, the multi-objective versions allow to generate a whole set of solutions, approximating the efficient frontier with a single run. Additionally, they do not require specialized software packages in order to solve quadratic programming problems or MIQPs. Access to highly efficient solvers of this type is a basic prerequisite of the approaches from Chapter 4 in order to get point-based approximation of the Pareto front. Unfortunately, commercial high-performance programs/libraries with the necessary capabilities are often quite expensive.

5.2 Multi-Objective Evolutionary Algorithms

In this section we will briefly introduce the main ideas behind multi-objective evolutionary algorithms (MOEAs) which are needed as foundation for the subsequent sections. For a more detailed introduction to MOEAs, the reader is referred to Deb [Deb01] or Coello et al. [CVL02].
Evolutionary algorithms are stochastic iterative optimization heuristics inspired by natural evolution. Starting with a set of candidate solutions (population), in each iteration (generation), promising solutions are selected as potential parents (mating selection), and new solutions (individuals) are created by mixing information from the parents (crossover) and slightly modifying them (mutation). The resulting offspring are then inserted into the population, replacing some old or less fit solutions (environmental selection). By continually selecting good solutions for reproduction and then creating new solutions based on the knowledge represented in the selected individuals, the solutions "evolve" and become better and better adapted to the problem to be solved, just like in nature, where the individuals become better and better adapted to their environment through the means of evolution.
The basic operations of an evolutionary algorithm can be described as follows:

1: $t \leftarrow 0$
2: initialize $P(0)$
3: evaluate $P(0)$
4: **while** (termination criterion not fulfilled) **do**
5: copy selected individuals into mating pool: $M(t) \leftarrow s(P(t))$
6: crossover: $M'(t) \leftarrow c(M(t))$
7: mutation: $M''(t) \leftarrow m(M'(t))$
8: evaluate $M''(t)$
9: update population: $P(t+1) \leftarrow u(P(t) \cup M''(t))$
10: $t \leftarrow t+1$
11: **end while**

with t denoting the generation counter, $P(t)$ the population at generation t, and s, c, m, and u representing the different genetic operators. Evolutionary algorithms have proven successful in a wide variety of applications. For a more detailed introduction to EAs, the reader is referred to Eiben and Smith [ES03].



(a) Non-dominance sorting to determine fronts (b) Crowding distance calculation within a front

Figure 5.1: NSGA-II first ranks individuals according to non-dominance sorting (a) and within a front according to crowding distance (b).

Because EAs maintain a population of solutions throughout the run, they can also be used to simultaneously search for a set of solutions approximating the efficient frontier of a multi-objective problem.
The main difference between single objective EAs and MOEAs is the way they rank their solutions for selection. While in single objective EAs the ranking is unambiguously defined by the solution quality, this is not so straightforward in the case of multiple objectives. MOEAs have two goals: they want to drive the population towards the efficient frontier, while at the same time maintaining a diverse set of alternative solutions. To achieve the first goal, most MOEA implementations rely on the concept of dominance. Solution A dominates solution B if A is at least as good as B in all objectives, and better in at least one objective. A solution is called *nondominated* with respect to a set of solutions if and only if it is not dominated by any other solution in that set.
One particularly popular MOEA variant is the nondominated sorting genetic algorithm (NSGA-II), see Deb [Deb01]. It ranks individuals according to two criteria. First, individuals are ranked according to non-dominancy: All nondominated individuals are assigned to Front 1, then they are removed from the population, and then again, all nondominated solutions are determined and are assigned Front 2, etc. until the population is empty. The result of this process is illustrated in Figure 5.1(a). Within a front, solutions are ranked according to the crowding distance, which is defined as the circumference of the rectangle defined by their left and right neighbors, and infinity if there is no neighbor. This concept is illustrated in Figure 5.1(b). Individuals with high crowding distance are preferred, as they are in more isolated regions of the objective space. In the example in Figure 5.1(b), individuals a and d have the highest priority within the front, followed by individual b and then c because the rectangle defined by the respective left and right neighbor is larger for individual b.
In every iteration, NSGA-II generates p offspring solutions, where p is the population size. The old population and the offspring are then combined, ranked according to the

above two criteria, and the better half forms the new population. For mating selection, tournament selection is used which randomly draws two solutions from the population and chooses the one on the better front or, if they are from the same front, the one with the larger crowding distance.

5.3 A Point-Based Multi-Objective EA

Although there is, to our knowledge, no definite comparison of the different approaches discussed in Section 5.1, we consider the approach by Streichert et al. [SUZ03, SUZ04a, SUZ04b] as state-of-the-art metaheuristic. Not only is it one of the few papers applying MOEAs, but they have also tested and compared a number of variations regarding encoding and operators. We re-implemented their algorithm, and will use it as reference for empirical comparison with our newly developed envelope-based MOEA. The algorithm is described below. Note that some adaptations were necessary to handle the 5-10-40-Constraint, as this constraint has not been considered in Streichert et al. [SUZ03, SUZ04a, SUZ04b].

We have already mentioned that the approach uses a MOEA for optimization. In particular, it is based on the standard nondominated sorting genetic algorithm (NSGA-II and its predecessor NSGA) as described above.

As genetic representation, Streichert et al. [SUZ03, SUZ04a, SUZ04b] recommend to use a hybrid binary/real-valued encoding, which is also used by several other successful approaches like Chang et al. [CMBS00]. With this encoding, a solution is defined by a vector of continuous variables $\boldsymbol{c} = (c_1, \ldots, c_N)^T$ representing the weights of the individual assets. An additional vector of binary variables $\boldsymbol{k} = (k_1, \ldots, k_N)^T$ is used to indicate if the asset is included in the portfolio at all. The latter vector allows the EA to easily add or remove an asset by simply flipping the corresponding bit, and thus facilitates the handling of cardinality constraints.

For a portfolio selection problem that contains a cardinality constraint and buy-in thresholds, the decoding of the two vectors to get the actual portfolio works as follows (see e.g. [CMBS00, SUZ04a]):

1. All c_i are set to zero if $k_i = 0$.
2. If $\sum_{i=1}^{N} \text{sign}(k_i c_i)$ is more or less than the required cardinality, the solution is repaired by changing some elements of $\boldsymbol{c}$. For the maximum cardinality problem, elements are set to 0 in the order of increasing c_i (i.e. the assets with the smallest share are set to zero).
3. The vector $\boldsymbol{c}$ is normalized:
$$c_i' = \frac{c_i}{\sum c_i} \tag{5.3}$$
4. The final weight x_i is calculated:
$$x_i = \begin{cases} l + c_i' \, (1 - |\Upsilon| l) & \text{if } i \in \Upsilon \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

where l is the minimum buy-in threshold and $|\Upsilon|$ denotes the number of elements in Υ.

The 5-10-40-Constraint has not yet been considered by Streichert et al., and thus we had to adapt the decoding and repair mechanism. It works in 7 steps as follows.

1. All c_i are set to zero if $k_i = 0$.
2. The vector $\boldsymbol{c}$ is normalized:
$$x_i = \frac{c_i}{\sum c_i}$$
3. The surplus amount exceeding the 10% threshold is calculated, $\Omega = \sum_{x_i > 0.1} (x_i - 0.1)$. All $x_i > 0.1$ are set to 0.1.
4. the surplus Ω has to be redistributed to the weights below 10%. Each weight below 10% is raised by the amount $(0.1 - x_i) \cdot \frac{\Omega}{\Psi}$, where $\Psi = \sum_{0 < x_i < 0.1} (0.1 - x_i)$. If $\Psi < \Omega$, no weight will have a value above 10% after the first step.
5. In the group with more than 5%, we only accept the assets with the largest weights such that the sum is still less than or equal to 40%. All others are capped to 5% and the excess weight is distributed to the other assets analogously to the above step.
6. If there is not enough room for all the excess weight to be redistributed among the assets with less than 5%, the remaining is used to fill up the assets between 5% and 10% up to 10% in order of decreasing weight.
7. The previous step may again lead to a violation of the 5-10-40-Constraint. In this case, assets are removed from the 5-10% group in order of increasing weights, and the weight in excess of 5% is distributed to the other assets in the 5-10% group similar to the previous step.

Note that any valid portfolio for the problem with 5-10-40-Constraint contains at least 16 different assets (4 times 10% plus 12 times 5%). If this is the case, the above decoding will result in a feasible portfolio. By appropriate mutation and crossover operators (see below) we make sure that the binary string has always at least 16 bits set to "1".
For both cardinality constraints and the 5-10-40-Constraint, the weights after the above repair steps are written back into the weight-vector of the genotype and overwrite the original values. This allows the information gained by the repair mechanism to be inherited to the offspring (Lamarckism) and has been recommended in Streichert et al. [SUZ04b].
For mutation, simple bit flip is used on the binary vector, and Gaussian mutation on the real-valued vector. For crossover, we use N-point crossover independently on both real-valued and binary vector. In Streichert and Ulmer [SUZ04a], this crossover operator was reported to be competitive to other, more complex crossover operators. We follow

this suggestion for the cardinality constrained problems. For the 5-10-40-Constraint, as explained above, we need at least 16 assets with $k_i = 1$ for a feasible solution. Therefore, for this constraint we modify the operators on the bit string as follows: The crossover operator first transfers all bits to the child where both parents are equal. The remaining bits are traversed in random order and randomly taken from either parent until the maximum number of zeros has been reached (i.e. $N - 16$ if N is the number of assets). Any remaining bits are set to 1. If, after mutation, the resulting bit string contains less than 16 ones, some of the performed 1 to 0 bit flips are reversed to make the string valid. In the remainder of this chapter, we will denote this point-based MOEA as P-MOEA.

5.4 An Envelope-Based Multi-Objective EA

In this section, we will present our new envelope-based multi-objective evolutionary algorithm (E-MOEA) for portfolio selection problems. It combines the efficiency of the PQP algorithm for calculating the whole continuous front with the ability of multi-objective evolutionary algorithms to take complex constraints into account and to generate multiple solutions within a single run. The main idea of the envelope-based approach is to use the MOEA to define suitable convex subsets of the original search space, run the PQP algorithm on every subset, and then recombine the partial solutions to form the complete front.

A single solution of the MOEA (i.e. an individual) defines a convex subset by specifying how the nonconvex constraints are to be handled. In case of a cardinality constraint, a solution defines which assets are allowed a weight greater than zero. The corresponding convex problem is just the standard problem which contains only those variables not forced to zero. Note that, in particular if the allowed cardinality is much smaller than the total number of available assets, this means that the generated sub-problem is much smaller than the original problem, which results in a tremendous reduction of the runtime of the PQP algorithm.

In the case of the 5-10-40-Constraint, a solution defines which assets are allowed up to 10% and hence have to be included in the 40% constraint. All other assets are restricted to at most 5%.

For each subset, the PQP algorithm can be used to efficiently calculate the whole Pareto front of the corresponding standard mean-variance portfolio selection problem. More details on the basic workings of the PQP algorithm are discussed in Chapter 3. Because we apply the PQP algorithm to every individual generated by the EA, an efficient implementation is crucial. We use a modified variant of the algorithm described by Best and Kale [BK00]. For an in-depth discussion of implementation intricacies, the reader is referred to Chapter 3 as well.

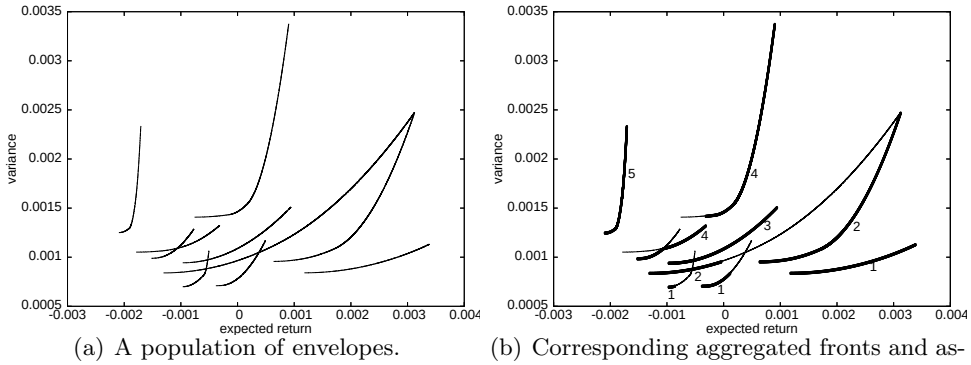
The result of the PQP algorithm is a front in the mean-variance space which is efficient for the sub-problem, but not necessarily for the overall problem with nonconvex constraints. We call such a partial front an **envelope**. The EA is now used to find a collection of such envelopes which together form a solution to the overall problem.

For this purpose, we use a multi-objective EA based also on the general framework of NSGA-II (see Deb [Deb01]). But instead of a solution being represented by a single point (portfolio) in the mean-variance space, now every solution is represented by an envelope in the mean-variance space. An exemplary population is depicted in Figure 5.2(a). The example shows that situations in which envelopes entirely dominate other envelopes occur very rarely. Instead, at many points, envelopes intersect with other envelopes. Even without intersections, envelopes may have dominated and nondominated parts. Thus, we had to adapt the nondominated sorting and crowding distance calculation to work with envelopes.
The basic idea can be described as follows:
We need to determine the *nondominated part of the set union of all envelopes* which will be called the (first) **aggregated front** in the remainder of this thesis. Following the idea of nondominated sorting, we assign Rank 1 to all individuals contributing at least partially to the first aggregated front. Then, we iteratively remove these individuals/envelopes from the population, and determine the aggregated front of the remaining individuals, assigning them the next higher rank, etc. The resulting ranking and the generated aggregated fronts are depicted in Figure 5.2(b).
It is clear that different individuals contribute differently to an aggregated front. Some may contribute only a small segment of the front, while others contribute large segments. Also, some parts of the aggregated front may be represented by several individuals. We use this information to rank the individuals within a front (substituting the crowding distance sorting in NSGA-II). For this purpose, we determine for each individual the length of the segment contributed to the aggregated front[2]. Parts common to several individuals are shared among those individuals. For example, if the part contributed solely by an individual i has length 5, and a part with length 4 is shared with another individual j, the overall contribution of individual i is $5 + 4/2 = 7$. Within a front, individuals contributing more are considered more important. Parts of the efficient frontier not belonging to the aggregated front are not taken into account.
The actual implementation of the above envelope-based nondominated sorting is more involved than one would assume at a first glance. The main part – the algorithm that calculates the aggregated front – will be briefly described below. For more details, the reader is referred to Scheckenbach [Sch06].

5.4.1 Calculating the Aggregated Front

The basic principle to determine the aggregated front from a given set of envelopes works as follows:
The algorithm starts with the envelope that contains the corner portfolio with the highest overall yield, since this portfolio – and thus the envelope it belongs to – is not dominated by any other envelope. Moving from this point in the direction of decreasing yield, the algorithm iteratively selects those envelopes that for a given value of $E(\boldsymbol{x})$

[2] Although in principle it would be possible to calculate the true length of a segment, for reasons of simplicity we approximated the length by the Euclidean distance between the end points. Another possible criterion would have been the reduction in hypervolume if the individual is removed.



(a) A population of envelopes.

(b) Corresponding aggregated fronts and assigned ranks.

Figure 5.2: Ten randomly initialized envelopes and the five corresponding aggregated fronts for a problem with a maximum cardinality constraint of $k = 4$ for the Nikkei dataset (P5).

dominate all the other envelopes. The collection of these nondominated parts represents the aggregated front.

In order to simplify the algorithm description, we will first introduce several basic equations and definitions:
Each envelope is determined by a collection of corner portfolios. Every portfolio $\boldsymbol{x}_\alpha$ in the interval between two adjacent corner portfolios $\boldsymbol{u}$ and $\boldsymbol{v}$ is defined as a convex combination of these two corner portfolios:

$$\boldsymbol{x}_\alpha = \alpha\boldsymbol{u} + (1-\alpha)\boldsymbol{v} = \boldsymbol{v} + \alpha(\underbrace{\boldsymbol{u}-\boldsymbol{v}}_{=:\boldsymbol{w}}), \quad \alpha \in [0,1] \tag{5.5}$$

Its expected return and variance are:

$$E(\boldsymbol{x}_\alpha) = \alpha E(\boldsymbol{u}) + (1-\alpha)E(\boldsymbol{v}) = E(\boldsymbol{v}) + \alpha E(\boldsymbol{w}), \quad \alpha \in [0,1] \tag{5.6}$$

$$V(\boldsymbol{x}_\alpha) = (\boldsymbol{v}+\alpha\boldsymbol{w})^T\boldsymbol{Q}(\boldsymbol{v}+\alpha\boldsymbol{w}) = \boldsymbol{v}^T\boldsymbol{Q}\boldsymbol{v} + 2\alpha\boldsymbol{v}^T\boldsymbol{Q}\boldsymbol{w} + \alpha^2\boldsymbol{w}^T\boldsymbol{Q}\boldsymbol{w}, \quad \alpha \in [0,1] \tag{5.7}$$

By using Eq. 5.6 to replace α in Eq. 5.7, V can be expressed solely in terms of $E(\boldsymbol{x}_\alpha)$:

$$\begin{aligned} V(E_\alpha) = {} & \boldsymbol{v}^T\boldsymbol{Q}\boldsymbol{v} - 2\boldsymbol{v}^T\boldsymbol{Q}\boldsymbol{w}\frac{E_v}{E_w} + \boldsymbol{w}^T\boldsymbol{Q}\boldsymbol{w} \\ & + 2\left(\frac{\boldsymbol{v}^T\boldsymbol{Q}\boldsymbol{w}}{E_w} - \frac{\boldsymbol{w}^T\boldsymbol{Q}\boldsymbol{w}}{E_w^2}\right)E_\alpha + \frac{\boldsymbol{w}^T\boldsymbol{Q}\boldsymbol{w}}{E_w^2}E_\alpha^2, \quad E_\alpha \in [E_u, E_v] \end{aligned} \tag{5.8}$$

(For sake of brevity, $E(\boldsymbol{x}_\alpha)$, $E(\boldsymbol{v})$, and $E(\boldsymbol{w})$ have been replaced by E_α, E_v, and E_w. We have further assumed that $E_u < E_v$.)

Eq. 5.8 demonstrates that in the mean-variance space, the image of each segment is part of a convex parabola, since

$$V''(E_\alpha) = \frac{\boldsymbol{w}^T \boldsymbol{Q} \boldsymbol{w}}{E_w^2} \geq 0$$

An envelope is therefore the continuous concatenation of parabola segments. When the aggregated front is considered, however, continuity usually can not be assumed, as e.g. Figure 5.2(b) shows.
In the further description of the procedure that calculates the aggregated front, the following expressions and abbreviations will be helpful:
The segment with the lowest expected return known to belong to the aggregated front will be called **current segment**, the envelope it is a part of will be referred to as the **current envelope**. The corner portfolio that marks the end of a segment and has the lowest expected return and variance is the **LCP**, the one with highest yield and variance is the **UCP**. The goal of each algorithm iteration is to find the next segment that is part of the aggregated front, the **succeeding segment**. The segment that was the current segment in the last iteration is referred to as the **preceding segment**.
There are three main scenarios how the transition from the current segment to the succeeding segment can take place:

1. The succeeding segment intersects the current segment.
2. The succeeding segment is reached by a vertical jump.
3. A horizontal or diagonal jump occurs at the end of the current segment.

The intersection scenario and the jump scenarios are treated separately:

Intersection

At the start, a list of candidates is compiled that comprises all segments that do not belong to the current envelope. Then, in the next step, segments are removed from the list if their exp. return coordinates don't overlap at all with exp. return coordinates of the current segment. We additionally used the convexity of the individual segments and of the whole envelope (cf. Markowitz [Mar87]) to further reduce the list size. For further details, the reader is referred to Scheckenbach [Sch06].
One possible way to calculate the intersection points is to equate the right hand sides of Eq. 5.8 for the current segment and the candidate segment.
Our procedure for the calculation is slightly different, however. Its main advantage is that it allows us to check easily for numerical errors when the succeeding segment has been chosen[3] (see Scheckenbach [Sch06]).
In a first step, we calculate "virtual" corner portfolios that shorten the two segments to be considered in way that they completely overlap with respect to their expected return

[3] If the slope of two segments that intersect is nearly identical, the numerical error can – as a consequence of limited floating point precision – be quite large. There is no simple way to cope with this besides checking regularly and making corrections if necessary.

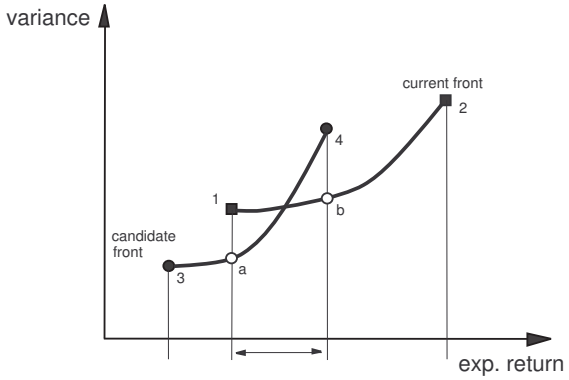


Figure 5.3: Creating virtual corner portfolios

coordinates. Suppose a situation as depicted in Figure 5.3, with the current segment defined by $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, and the candidate segment determined by $\boldsymbol{x_3}$ and $\boldsymbol{x_4}$. Then the virtual portfolios $\boldsymbol{x_a}$ and $\boldsymbol{x_b}$ are calculated as follows:

$$\boldsymbol{x_a} = \boldsymbol{x_3} + \frac{E(\boldsymbol{x_1}) - E(\boldsymbol{x_3})}{E(\boldsymbol{x_4}) - E(\boldsymbol{x_3})}(\boldsymbol{x_4} - \boldsymbol{x_3})$$

$$\boldsymbol{x_b} = \boldsymbol{x_1} + \frac{E(\boldsymbol{x_4}) - E(\boldsymbol{x_1})}{E(\boldsymbol{x_2}) - E(\boldsymbol{x_1})}(\boldsymbol{x_2} - \boldsymbol{x_1})$$

In the virtual segments $(\boldsymbol{x_a} \rightarrow \boldsymbol{x_4})$ and $(\boldsymbol{x_1} \rightarrow \boldsymbol{x_b})$, the portfolios at an intersection can be described as in Eq. 5.5:

$$\boldsymbol{x}_{\text{candidate}} = \boldsymbol{x_a} + t_{\text{candidate}}(\boldsymbol{x_4} - \boldsymbol{x_a}),\ t_{\text{candidate}} \in [0, 1]$$

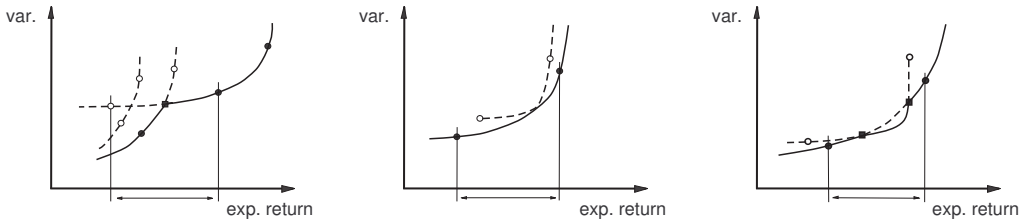
$$\boldsymbol{x}_{\text{current}} = \boldsymbol{x_1} + t_{\text{current}}(\boldsymbol{x_b} - \boldsymbol{x_1}),\ t_{\text{current}} \in [0, 1]$$

Since the intervals $[E(\boldsymbol{x_a}), E(\boldsymbol{x_4})]$ and $[E(\boldsymbol{x_1}), E(\boldsymbol{x_b})]$ are identical, $t_{\text{candidate}}$ and t_{current} are the same at an intersection point and can therefore be replaced by t. With this, equating the variances at the intersection point leads to:

$$\begin{aligned}\boldsymbol{x_a}^T\boldsymbol{Q}\boldsymbol{x_a} + 2\left(\boldsymbol{x_4} - \boldsymbol{x_a}\right)^T\boldsymbol{Q}\boldsymbol{x_a} \cdot t + \left(\boldsymbol{x_4} - \boldsymbol{x_a}\right)^T\boldsymbol{Q}\left(\boldsymbol{x_4} - \boldsymbol{x_a}\right) \cdot t^2 = \\ \boldsymbol{x_1}^T\boldsymbol{Q}\boldsymbol{x_1} + 2\left(\boldsymbol{x_b} - \boldsymbol{x_1}\right)^T\boldsymbol{Q}\boldsymbol{x_1} \cdot t + \left(\boldsymbol{x_b} - \boldsymbol{x_1}\right)^T\boldsymbol{Q}\left(\boldsymbol{x_b} - \boldsymbol{x_1}\right) \cdot t^2\end{aligned}$$

$$\Longleftrightarrow$$

$$\begin{aligned}&\left[\left(\boldsymbol{x_4} - \boldsymbol{x_a}\right)^T\boldsymbol{Q}\left(\boldsymbol{x_4} - \boldsymbol{x_a}\right) - \left(\boldsymbol{x_b} - \boldsymbol{x_1}\right)^T\boldsymbol{Q}\left(\boldsymbol{x_b} - \boldsymbol{x_1}\right)\right] \cdot\ t^2 \\ &\quad + 2\left[\left(\boldsymbol{x_4} - \boldsymbol{x_a}\right)^T\boldsymbol{Q}\boldsymbol{x_a} - \left(\boldsymbol{x_b} - \boldsymbol{x_1}\right)^T\boldsymbol{C}\boldsymbol{x_1}\right] \cdot\ t + \left[\boldsymbol{x_a}^T\boldsymbol{Q}\boldsymbol{x_a} - \boldsymbol{x_1}^T\boldsymbol{Q}\boldsymbol{x_1}\right] = 0 \quad (5.9)\end{aligned}$$



(a) Several envelopes intersecting at different points.

(b) Two envelopes touching but not intersecting.

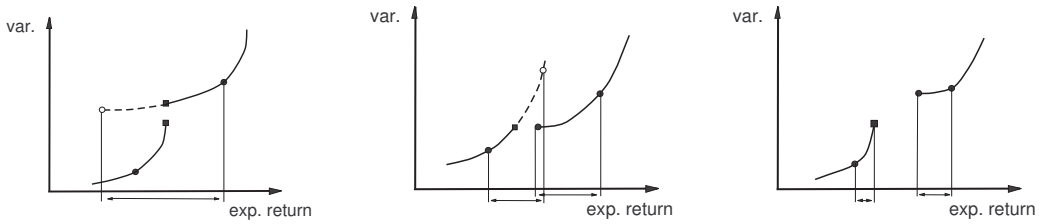
(c) Two envelopes intersecting twice.

Figure 5.4: Different intersections constellations

This quadratic equation can have two solutions, or one, or none at all. If there is no solution, the parabolas of the two segments don't intersect. In the rare case of one solution, the candidate segment touches the current segment (see Fig. 5.4(b)) but remains dominated. When there are two solutions, i.e. there is a "real" intersection, a solution is only relevant if it is in $[0, 1]$, as only these result in an intersection in the interval $[E(\boldsymbol{x_a}), E(\boldsymbol{x_b})]$. Usually, at most one of the two solutions will result in a feasible solution in the interval (Fig. 5.4(a)), but in rare cases, the two segments intersect twice in $[E(\boldsymbol{x_a}), E(\boldsymbol{x_b})]$ (see Fig. 5.4(c)). Then the intersection point with the higher expected return has to be chosen.
When the complete list of candidates has been covered, the intersection point with the highest expected return in the interval $[E(\text{LCP}), E(\text{UCP})]$ of the current segment defines the succeeding envelope, but only if there is no vertical jump point with an even higher expected return.

Jump Points

A vertical jump to another envelope (Figure 5.5(a)) can occur everywhere on the current segment, but the succeeding segment has to be situated at the end of another envelope, namely the one with the highest expected return. It is therefore sufficient to check if the high-yield end of every envelope is in the interval $[E(\text{LCP}), E(\text{UCP})]$ of the current segment. Of all those envelopes fulfilling the condition, only those that start below the current segment dominate part of the current segment. Of these dominating envelopes, the one that starts with the highest expected return determines the succeeding envelope if no intersection with a higher expected return has been found.
A horizontal jump (Figure 5.5(b)) can only occur when the minimum variance portfolio (MVP) of an envelope is reached, i.e. when the current envelope ends and neither an intersection point nor a suitable vertical jump is found in the last segment. In this case all segments of the other envelopes are identified that have a UCP with higher variance and an LCP with lower variance than the variance of the MVP at the end of the current envelope. Of these candidate segments only those are viable to be the succeeding segment that have an LCP with a lower expected return than the MVP of the current envelope.



(a) Vertical jump to another envelope.

(b) Horizontal jump at the end of an envelope.

(c) Diagonal jump at the end of an envelope.

Figure 5.5: Possible jump discontinuities within the current mean interval

For each of the remaining candidates an artificial corner portfolio with the same variance as the MVP is constructed, and the one with the highest expected return is the candidate that is chosen as succeeding segment. Before the succeeding segment is definitely chosen, however, we have to check if a diagonal jump (Figure 5.5(c)) occurs that dominates the segment of the horizontal jump. For this to happen, there must be an envelope with a maximum exp. return smaller than that of the MVP and larger than the one of the artificial corner portfolio of the chosen candidate. Additionally, the variance of this envelope must be lower than the variance of the artificial corner portfolio. Is there more than one envelope that fulfills the conditions for a diagonal jump, the one that has the highest expected return is chosen.
When such a diagonal jump happens, it has to be recorded when the aggregated front is stored. Otherwise it can not be easily detected that there is a discontinuity in the front. (Horizontal and vertical jumps can be detected easily from the corner portfolios of the aggregated front, as there are two consecutive portfolios with either no change in variance or expected return.)
What was not covered above were cases that are the result of overlapping envelopes. For more details on how these are handled, the reader is referred to Scheckenbach [Sch06].
The whole procedure to assemble the aggregated front terminates when the portfolio with the lowest variance of all envelopes is reached.

5.4.2 Representation and Genetic Operators

As discussed above, the EA is only responsible for handling the nonconvex constraints, the appropriate weights are then determined by a PQP algorithm. Thus, in principle, a simple binary encoding would be sufficient. However, we wanted to feed back some information from the PQP algorithm to the evolutionary algorithm. For this reason, we are using a permutation encoding. Then, for the maximum cardinality constraint, simply the first K assets are used in the portfolio. For the 5-10-40-Constraint, the first K assets are considered potential heavyweights, with a share of at most 10%[4] and inclusion

[4]Note that the weight of these assets can be set below 5% by the PQP algorithm, although they are still included in the 40% constraint. This helps in the sorting, as such an asset is then moved behind more important assets that are set to 5%, reducing the chance to be included again after mutation.

in the 40% constraint, while all others are restricted to less than 5%. The parameter K here is variable and also part of the solution encoding.
After the PQP algorithm has been applied, the permutation is sorted with respect to the average weight an asset had in all corner portfolios that are part of the aggregated front. Thus, an asset which received a high weight will appear early on the permutation, and subsequently have a higher probability to be among the first K after crossover and mutation.
As genetic operators, we use the uniform order based crossover and swap mutation. For the latter, an asset that belongs to the first K is swapped with an arbitrary other asset. For the 5-10-40-Constraint, the parameter K is modified by adding a Gaussian number with mean 0 and standard deviation P_m. The size is then rounded and capped if necessary.
To further improve the efficiency of our algorithm, we introduced two additional concepts: duplicate elimination and a variable population size. In duplicate elimination, we remove individuals that share a part of an aggregated front with another individual, but which are nowhere better than the other individual. The variable population size allows us to increase the number of individuals in the population if the current first aggregated front consists of more individuals than would fit into the population. Keeping a fixed population size would then mean to delete a valuable part of the solution. Note that because our approach is envelope-based, it requires a much smaller population size than point-based approaches anyway. The alternative to an adaptive population size, namely to work with an equally large population as the point-based approaches from the beginning, would have slowed down convergence unnecessarily. Independent of the population size, the number of offspring generated in every iteration remains constant and equal to the original population size.

5.5 Empirical Evaluation

To test the new approach, we used three of the benchmark data sets from the OR-Library [Bea06], P1, P4, and P5. Because we want to show that our algorithm also scales well, we additionally tested it on P6. (For further details on the benchmarks, the reader is referred to Section 2.4.)
For the cardinality constrained problems, we set the maximum cardinality to $K = 4$ for P1 and P4, and to $K = 8$ for P5 and P6.
To measure algorithm performance we again provide both the max. delta-area and the ideal delta-area (see Section 4.1 for a detailed explanation).

5.5.1 Parameter Settings

For P-MOEA, we use the same parameter settings as in Streichert and Ulmer [SUZ04a], i.e. a population size of 250 and tournament size of 8. For the bit string, bit flip mutation with mutation probability for each bit 2/(number of assets) and N-point crossover with probability 1.0 are applied. For the real-valued string, mutation is done by adding a

value from a normal distribution with $\sigma = 0.05$ to each weight, crossover is again N-point crossover with probability 1.0.
For E-MOEA, the following parameter settings have been chosen without much testing. The initial population size is set to 30, and 30 individuals are generated in every iteration. The mutation and crossover operators have been described above. Probability to swap each of the first K assets for the cardinality problem is $1/K$, for problems with 5-10-40-Constraint, probability to swap any of the heavyweights is 1/7.
As discussed before, P-MOEA requires a significantly larger population size, as each individual only represents a single point, as opposed to a whole envelope as in E-MOEA. All reported results are averages over 30 runs. Experiments were conducted on a PC with AMD Sempron 1.6 GHz processor and 1 GB RAM. Maximum allowed running time for the problem with 5-10-40-Constraint was set to 500, 1000, 2000, and 4000 CPU seconds for P1, P2, P3, and P4, respectively. Since the cardinality constrained problem seemed easier, we allowed only half the running time for each problem.

5.5.2 Test Results

The results on all 4 benchmark problems are summarized in Tables 5.1 to 5.4. Table 5.1 reports on the max. delta-area, i.e. the area between obtained efficient frontier and ideal front with wide margins, for the cardinality constrained problem at the end of the run. The same information, but with respect to ideal delta-area, is provided in Table 5.2. As can be seen, E-MOEA significantly outperforms P-MOEA on all benchmark problems. In terms of the ideal delta-area, the relative performance of P-MOEA is somewhat better, indicating that it particularly has a problem in finding the portfolios with high expected return or low variance.

Table 5.1: Max. delta-area at the end of the run for E-MOEA and P-MOEA on test problems with cardinality constraint, average ± std. error, all values in terms of 10^{-6}.

	P1 ($K = 4$)	**P4** ($K = 4$)	**P5** ($K = 8$)	**P6** ($K = 8$)
P-MOEA	**1.1613**	**2.7787**	**9.3292**	**4124.1363**
standard error	± 0.0159	± 0.0521	± 0.2287	± 123.6716
E-MOEA	**0.2275**	**0.8048**	**0.0561**	**5.9063**
standard error	± 0	± 0.00003	± 0.0052	± 0.2939

The results for the problem with 5-10-40-Constraint look similar (see Tables 5.3 and 5.4), although the differences between P-MOEA and E-MOEA are generally smaller.
Typical efficient frontiers for P1 and P5 with cardinality constraint are depicted in Figure 5.6. As can be seen, for the small problem (P1), both algorithms perform quite well. In fact, the figure zooms in on only a part on the front, as on a plot of the whole front, the differences would be hard to see. Still, E-MOEA clearly outperforms P-MOEA and is indistinguishable from the ideal front over large parts. For the larger

Table 5.2: Ideal delta-area at the end of the run for E-MOEA and P-MOEA on test problems with cardinality constraint, average ± std. error, all values in terms of 10^{-6}.

	P1 ($K = 4$)	**P4** ($K = 4$)	**P5** ($K = 8$)	**P6** ($K = 8$)
P-MOEA	**1.0605**	**1.6981**	**2.1250**	**125.1611**
standard error	± 0.0160	± 0.0175	± 0.0189	± 1.1580
E-MOEA	**0.1371**	**0.5222**	**0.0123**	**2.4398**
standard error	± 0	± 0.00003	± 0.0003	± 0.09637

Table 5.3: Max. delta-area at the end of the run for E-MOEA and P-MOEA on test problems with 5-10-40-Constraint, average ± std. error, all values in terms of 10^{-6}.

	P1	**P4**	**P5**	**P6**
P-MOEA	**1.3274**	**1.5107**	**1.7339**	**446.7284**
standard error	± 0.0002	± 0.0197	± 0.0138	± 11.4963
E-MOEA	**1.3019**	**1.2416**	**1.6511**	**344.7926**
standard error	± 0.0007	± 0.0001	± 0.0047	± 3.8489

problem, P-MOEA does not seem to be able to come close to the performance of E-MOEA. In particular in the area of higher returns, its deficiencies are obvious. It seems that P-MOEA does not scale very well to larger problem sizes. (In Streichert et al. [SUZ04a, SUZ04b, SUZ03], the algorithm was only tested on small problems with up to 81 assets.) One reason may be that there are usually only few assets with a high return, and exactly those have to be combined in the portfolio to obtain an overall high return. Identifying the high-return assets out of a large set may prove difficult for the P-MOEA. E-MOEA on the other hand finds solutions hardly distinguishable from the ideal front also for the larger problems.

Table 5.4: Ideal delta-area at the end of the run for E-MOEA and P-MOEA on test problems with 5-10-40-Constraint, average ± std. error, all values in terms of 10^{-6}.

	P1	**P4**	**P5**	**P6**
P-MOEA	**0.2188**	**0.1028**	**0.0807**	**7.6607**
standard error	± 0.00002	± 0.0009	± 0.0003	± 0.1116
E-MOEA	**0.2023**	**0.0631**	**0.0700**	**4.9756**
standard error	± 0.00001	± 0.0001	± 0.00006	± 0.0290

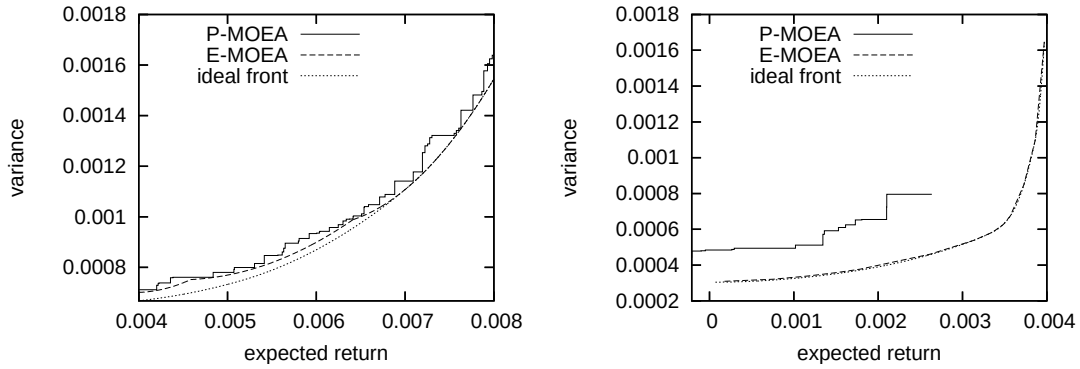


Figure 5.6: Typical fronts obtained on test problem P1 (left) and test problem P5 (right) with cardinality constraint.

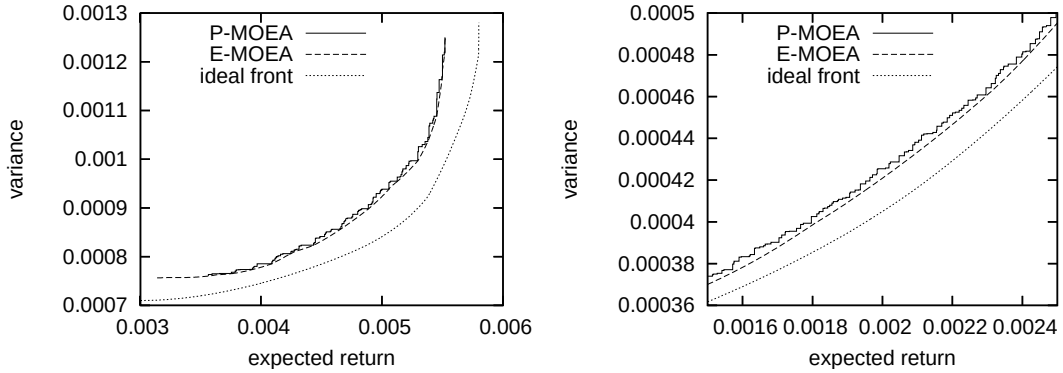


Figure 5.7: Typical fronts obtained on test problem P1 (left) and test problem P5 (right) with 5-10-40-Constraint.

For the 5-10-40-Constraint, the fronts obtained by P-MOEA and E-MOEA are much closer to each other, and further away from the ideal front. Still, the front obtained by E-MOEA dominates P-MOEA's front basically everywhere. Note that again for visibility, the plot for the larger problem only shows a segment of the overall front.
When looking at the obtained solution quality in terms of max. delta-area over running time, it is clear that the advantage of E-MOEA over P-MOEA is significant throughout the run. Figure 5.8(a) looks at P5 with cardinality constraint. Clearly, E-MOEA starts out much better, and converges much faster than P-MOEA (for the small problem, E-MOEA even found the best solution within 5 out of the 250 available seconds in every single run). Note that we plot against running time. Because E-MOEA has to run a PQP algorithm for every individual, it can only evaluate about 13500 individuals during the 1000 seconds allowed, while P-MOEA generates and evaluates approximately 1,945,000 individuals in the same time frame.

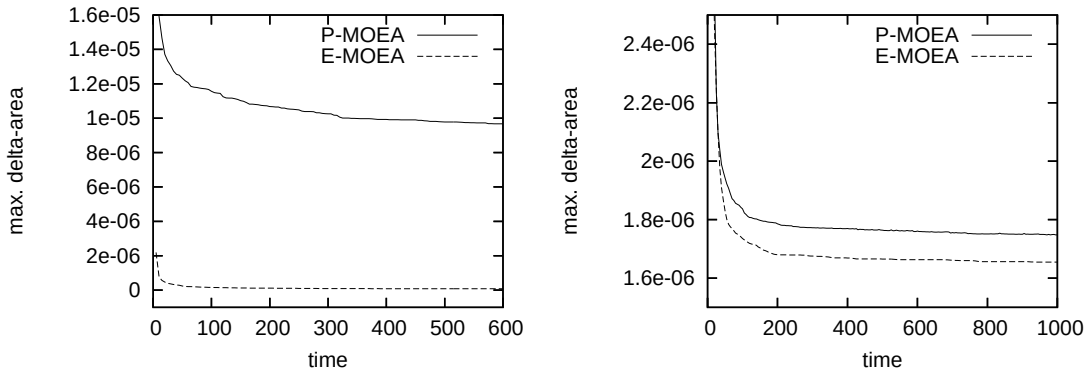


Figure 5.8: Convergence curves for P5 with cardinality constraint (left) and 5-10-40-Constraint (right).

For the problem P5 with 5-10-40-Constraint, E-MOEA also starts better than P-MOEA, then P-MOEA quickly catches up only to fall behind again (see Figure 5.8(b)). The difference in the number of individuals evaluated is even more striking here than for P5 with cardinality constraint, since the 5-10-40-Constraint does not allow to remove a large fraction of the assets for the PQP algorithm. While E-MOEA can generate only about 3000 individuals in the given 2000 seconds, P-MOEA generates approximately 3,475,000.
One explanation for the superiority of E-MOEA, besides being envelope-based, is certainly its in-built weight optimization by the PQP algorithm. This effect is visible in Figure 5.9, which depicts the solution quality of randomly generated solutions for both the P-MOEA and the E-MOEA. Clearly, E-MOEA has a much better start, as the majority of randomly generated envelopes is clearly better than the majority of randomly generated portfolios.

5.6 Concluding Remarks

Parametric quadratic programming algorithms – like the one described in detail in Chapter 3– are a very efficient way to calculate the whole efficient frontier for a standard mean-variance portfolio selection problem. The inclusion of nonconvex constraints such as a maximum cardinality constraint, buy-in thresholds, or the 5-10-40 rule from the German investment law, however, renders the feasible region nonconvex and thus prevents us from applying this fast and efficient method. In the previous chapter we have therefore – similar to other researchers – resorted to solving these problems point-based. We approximated the efficient frontier by iteratively solving a sub-problem with a fixed expected return E_f, and repeated this for many different settings of E_f. Most other publications work within a meta-heuristic framework, but nevertheless, the approaches are all point-based.

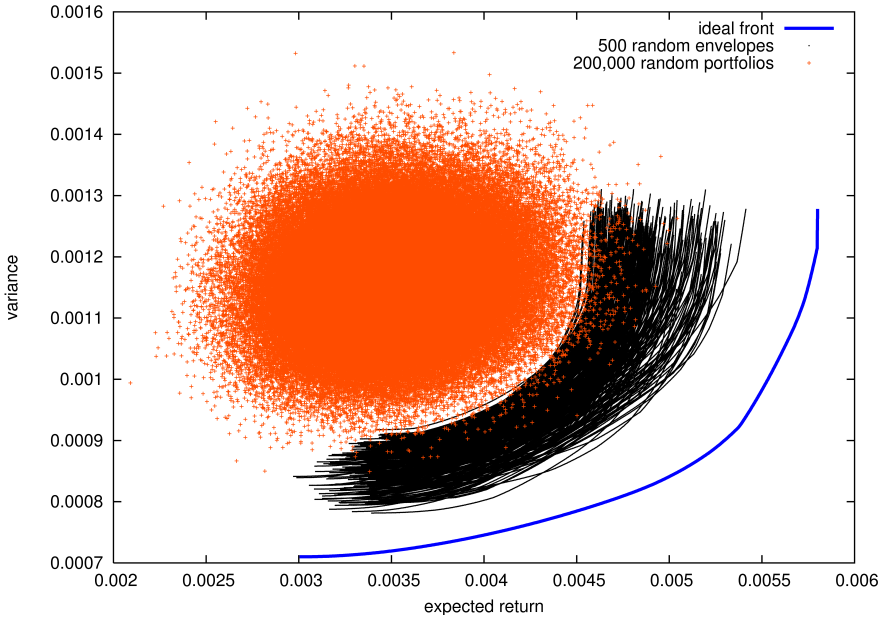


Figure 5.9: Randomly initialized populations of envelopes and portfolios for the 5-10-40 problem and the Hang Seng Dataset (P1).

In this chapter, we have proposed a new envelope-based multi-objective evolutionary algorithm (E-MOEA), which is a combination of a multi-objective algorithm with an embedded solver for parametric quadratic programming. The task of the MOEA is to define a set of convex subsets of the search space. Then, for each subsets the PQP algorithm can efficiently generate an efficient frontier, a so-called envelope. The combination of all generated envelopes then forms the overall solution to the problem.
To our knowledge, our approach is the first metaheuristic approach which is not point-based, but which is capable of generating a continuous front of alternatives for portfolio selection problems with nonconvex constraints. Compared with a state-of-the-art point-based MOEA, E-MOEA was shown to find significantly better frontiers in a shorter time. Both approaches presented in this chapter do not depend on the availability of a highly efficient QP solver, as did the ϵ-Constraint heuristics presented in the previous chapter.
The results presented above showed clearly that the E-MOEA is superior to the P-MOEA with respect to solution quality and calculation time. The only arguments for P-MOEA are flexibility and ease of implementation: (i) When working with the P-MOEA, it is not too problematic to exchange the variance as a measure of dispersion with e.g. Value at Risk. (ii) It is by far easier to implement an algorithm similar to the P-MOEA we have used, but it definitely is not trivial to produce an working version of the E-MOEA, since this additionally requires a fast PQP algorithm and an efficient procedure to calculate the aggregated front.

As future work, we are planning to integrate a more intelligent mutation operator that uses shadow prices to influence mutation probabilities. Also, the ideas of envelope-based MOEAs should be transferred to other applications. In particular, the idea seems very promising when applied to portfolio re-balancing problems that include fixed and variable transaction costs.

6 Combining Point-Based and Envelope-Based Approaches

In the previous chapter, we introduced a new type of algorithm for solving nonconvex portfolio selection problems, the envelope-based multi-objective evolutionary algorithm (E-MOEA). The core idea of this new approach was to have individuals that not only describe one point in the solution space but instead define a whole area of the Pareto front. This can be achieved by using a set of convex constraints as a new encoding scheme for an individual.
The cost – measured in computation time – for "decoding" such an individual, i.e. for calculating the envelope, is usually much higher than the construction of a single portfolio by e.g. a point-based MOEA. It stands to reason that heuristic knowledge might be beneficial in determining which sets of convex constraints are to be used for the envelopes instead of the stochastic search we applied in Chapter 5. The simple idea for an enhanced algorithm is therefore to gain this knowledge from feasible points that have been constructed before, with either point-based heuristics or by the application of a mixed integer solver for quadratic programs.
In the following section, we will describe how this notion has been realized and which design decisions have been taken in the process. We then present test results documenting the performance of this new algorithm in Section 6.2.

6.1 Algorithm Description

Provided a feasible point in the search space, it is a simple task – at least for the problem types considered here – to determine a convex set of linear constraints that (i) comply with all given constraints, both convex and nonconvex and that (ii) define the envelope which the respective portfolio is part of.
The *combination algorithm* we have designed can be divided into four main parts:

1. The calculation of a limited number of feasible solutions with either an MIQP solver (guarantees optimality, needs more time) or any of the presented ϵ-Constraint heuristics (fast, but solution quality may be lacking).

2. The analysis of the feasible solutions to determine a set of convex constraints for each. These sets are then added to the collection of sets, but only if an identical set is not yet present in the collection.

3. For each of the sets in the collection the respective envelope is calculated.

4. All envelopes are accumulated to the aggregated front with the algorithm presented in Section 5.4.

We will discuss the first two parts in more detail, the third and fourth are identical to the procedure we used as part of the envelope-based MOEA (Section 5.4):

Calculating a Set of Nondominated Points

Both the results of the heuristics and the results of the mixed-integer models presented in Chapter 4 can be used as feasible solutions – as could the results of the point-based MOEA. Due to the large amount of computation time required to achieve a sufficiently good solution with a P-MOEA, it does not make sense to use this kind of heuristic as a basis.
We have made the observation that the aggregated front for portfolio selection problems with nonconvex constraints usually consists of only very few envelopes, typically between 5 and 20. In order to construct the actual Pareto front, at least one feasible point on every envelope would have to be known, as one point is enough to determine the convex set of constraints for the envelope which the point is a part of. Unfortunately, the placement of the envelopes that are part of the Pareto front is not known at the time when the expected return values of the points have to be chosen. Thus enough points have to be distributed along the axis of expected return, and we have to place them in a way that has a high probability for each of the envelopes of the Pareto front to be "touched" at least once.
Depending on whether higher speed or better solution quality is desired, we have to choose a suitable single point algorithm from the ones presented in Chapter 4. If fast computation is more important than being on or near the optimum, one of the heuristics is the better choice, since the time spent on each individual point is by far smaller and varies less. Is it, however, more important to get a solution as close as possible to the actual Pareto front, the MIQP solver is clearly the algorithm we would select.
As it is not obvious which of the two distribution schemes, equidistant placement or the 2-Phase procedure, is better in this context, we tested both variants. The main advantage of the 2-Phase procedure – the placement of the points in a way that reduces the area between the nondominated front and the ideal front – does probably not come into play here, since the calculated portfolios don't exist as actual points in the final solution but are instead just used to determine the envelopes.

Determining the Sets of Convex Constraints

Given a point that is on the actual Pareto front (if an MIQP solver is used in the first part of the algorithm) or close to it (in case of the heuristics), the set of convex constraints that defines an envelope can be easily determined by analyzing such a point:

- For problems with 5-10-40-Constraint, we only have to set the upper bounds of those assets with a weight larger than $0.05 + \epsilon$ to 10%[1], and their sum has to be capped by 40%. The upper bound of all the other assets is set to 5%.

[1]The introduction of a very small positive constant ϵ is necessary in order to account for rounding errors and limited precision of the QP solver.

- For problems with a maximum cardinality constraint, the assets that are smaller than $0+\epsilon$ are set to equal 0 and can be removed completely from the problem data (covariance matrix, vector of expected returns) when the envelope is calculated.
- Similarly, for problems with buy-in thresholds, all assets that are smaller than $0+\epsilon$ are set to 0 as well, but additionally the lower bound of those assets with a weight larger than $0+\epsilon$ is set to the threshold.

After this has been done for all available points, the number of envelopes to be calculated can be significantly reduced by eliminating duplicates. This is achieved by starting with an empty collection of convex sets and consecutively adding new sets of convex constraints based on the calculated points, whereas a new set of constraints is only included in the collection if an identical set is not yet present in the collection. For each member of the collection, the respective envelope is calculated, and, in a last step, the aggregated front is determined by combining the thus constructed envelopes.

6.2 Test Results

The following test results demonstrate how much the combination algorithm reduces calculation time and improves solution quality in comparison to the point-based heuristics. In these tests, we only considered problems with either 5-10-40-Constraint or maximum cardinality constraint[2].
As test platform, we once more used a computer with an AMD-Athlon-CPU with 1,5 GHz and Linux (Debian sid) as operating system. The program was compiled with version 3.3 of the Gnu Compiler Collection [GCC06] with optimization level 3.
As solver for the mixed integer and the "standard" quadratic programming part, CPLEX 9.0 from ILOG [CPL06] was chosen again.
We tested both the heuristics and the MIQP solver for the calculation of the individual points. But we reduced the point budget from 400 to 40, which we assumed should still be enough to reach each of the nondominated envelopes at least once. These 40 points were distributed either equidistantly or with the 2-Phase procedure (cf. Section 4.2), and we report the results for both approaches. When we applied the 2-Phase distribution scheme, we divided the 40 points with a 1:3 ratio on the first and second phase, and not with the 1:9 ratio we used with the 400 point budget. The reason for this modification was to ensure that enough points were allocated in the first phase to have a good chance for the distribution algorithm to be able to work reliably.
For problem instances with 5-10-40-Constraint, Table 6.1 describes the results we obtained when we used the 2-solver-calls heuristic and the n-solver-calls heuristic in the first part of the combination algorithm[3]. For each configuration (i.e. each problem instance, each of the heuristics, and the two distribution schemes), the values for the max. delta-area and the ideal delta-area are listed.

[2] The algorithm for buy-in thresholds is nearly identical to the maximum cardinality case.
[3] As the partial derivatives heuristic performed poorly, we had decided to limit the tests to the other two heuristics.

Table 6.1: Test results for the combination algorithm on problems with 5-10-40-Constraint. The 40 points in the first part were generated by the 2-solver-calls and the n-solver-calls heuristic, with results for equidistant spacing and 2-Phase spacing of the points reported separately. The reduction of the max. delta-are, ideal delta-area, and CPU-time in comparison to the "simple" ϵ-Constraint based heuristics with 400 points and 2-Phase distribution is reported in parentheses. Algorithm runtime was measured in seconds.

	2-solver-calls		**n-solver-calls**	
	equal distr.	**2-Phase**	**equal distr.**	**2-Phase**
P1 ($N = 31$)				
ideal delta-area (red.)	2.07e-07 (68%)	2.04e-07 (69%)	2.08e-07 (29%)	2.08e-07 (29%)
max. delta-area (red.)	1.41e-06 (56%)	1.39e-06 (57%)	1.40e-06 (46%)	1.40e-06 (46%)
CPU-time (red.)	0.24 (88%)	0.24 (88%)	0.28 (90%)	0.31 (89%)
P2 ($N = 85$)				
ideal delta-area (red.)	3.56e-08 (64%)	3.51e-08 (64%)	3.57e-08 (42%)	3.57e-08 (42%)
max. delta-area (red.)	1.19e-06 (46%)	1.21e-06 (45%)	1.21e-06 (38%)	1.21e-06 (38%)
CPU-time (red.)	0.71 (82%)	0.73 (82%)	1.03 (83%)	0.97 (84%)
P3 ($N = 89$)				
ideal delta-area (red.)	2.40e-08 (63%)	2.26e-08 (66%)	1.93e-08 (45%)	1.93e-08 (45%)
max. delta-area (red.)	6.99e-07 (37%)	6.46e-07 (42%)	4.76e-07 (42%)	4.76e-07 (42%)
CPU-time (red.)	0.78 (79%)	0.8 (78%)	1.14 (80%)	1.19 (79%)
P4 ($N = 98$)				
ideal delta-area (red.)	6.34e-08 (59%)	6.34e-08 (59%)	6.34e-08 (31%)	6.27e-08 (32%)
max. delta-area (red.)	1.24e-06 (42%)	1.24e-06 (42%)	1.24e-06 (27%)	1.24e-06 (27%)
CPU-time (red.)	1.48 (61%)	1.68 (55%)	1.64 (70%)	2.07 (63%)
P5 ($N = 225$)				
ideal delta-area (red.)	8.17e-08 (62%)	7.82e-08 (64%)	7.74e-08 (21%)	7.63e-08 (23%)
max. delta-area (red.)	2.09e-06 (37%)	2.06e-06 (38%)	1.97e-06 (19%)	1.97e-06 (19%)
CPU-time (red.)	1.89 (84%)	2.39 (79%)	3.49 (85%)	4.15 (82%)
P6 ($N = 500$)				
ideal delta-area (red.)	5.16e-06 (54%)	5.15e-06 (54%)	5.89e-06 (5%)	4.79e-06 (23%)
max. delta-area (red.)	3.76e-04 (39%)	3.76e-04 (39%)	4.54e-04 (4%)	3.11e-04 (34%)
CPU-time (red.)	30.37 (39%)	37.12 (25%)	33.5 (68%)	50.84 (51%)
P7 ($N = 1000$)				
ideal delta-area (red.)	8.39e-06 (58%)	8.34e-06 (58%)	8.36e-06 (32%)	8.30e-06 (32%)
max. delta-area (red.)	1.46e-03 (32%)	1.46e-03 (32%)	1.46e-03 (29%)	1.46e-03 (29%)
CPU-time (red.)	98.27 (44%)	114.37 (35%)	114.15 (62%)	182.62 (39%)

The relative reduction of these areas in comparison the results we got by just applying the purely point-based ϵ-Constraint approaches (with a budget of 400 points) is printed in parentheses. Additionally we report on the runtime of the algorithms. The percentage of time the combination algorithm variants saved in comparison to the runtime of the "normal" ϵ-Constraint heuristics (again with a 400 point budget) is reported in parentheses as well.

Table 6.2 does the same for the reoptimization heuristic and the n-solver-calls heuristic for problem instances with a maximum cardinality constraint. As in previous tests, the cardinality K was set to 4 for Problems 1–4, and to 8 for Problems 5–7.
Finally, the results for the combination of the MIQP solver with the envelope approach are listed in Table 6.3, for problems with 5-10-40-Constraint and for the instances with maximum cardinality constraint.

Table 6.2: Test results for the combination algorithm on problems with a maximum cardinality constraint. The 40 points in the first part were generated by the reoptimization heuristic and the n-solver-calls heuristic, with results for equidistant spacing and 2-Phase spacing of the points reported separately. The reduction of the max. delta-are, ideal delta-area, and CPU-time in comparison to just applying the "simple" heuristics with 400 points and 2-Phase distribution is reported in parentheses. The maximum cardinality was chosen as in all previous tests: $K = 4$ for P1–P4, $K = 8$ for P5–P7. Algorithm runtime was measured in seconds.

	reoptimization		**n-solver-calls**	
	equal distr.	**2-Phase**	**equal distr.**	**2-Phase**
P1 ($N = 31$)				
ideal delta-area (red.)	1.38e-07 (29%)	1.51e-07 (22%)	1.43e-07 (26%)	1.51e-07 (21%)
max. delta-area (red.)	2.29e-07 (20%)	2.44e-07 (14%)	2.33e-07 (18%)	2.44e-07 (14%)
CPU-time (red.)	0.09 (71%)	0.07 (77%)	0.13 (71%)	0.09 (80%)
P2 ($N = 85$)				
ideal delta-area (red.)	5.10e-07 (24%)	5.10e-07 (24%)	4.99e-07 (3%)	5.01e-07 (2%)
max. delta-area (red.)	8.89e-07 (15%)	8.89e-07 (15%)	8.78e-07 (1%)	8.80e-07 (1%)
CPU-time (red.)	0.29 (80%)	0.19 (87%)	1.05 (77%)	0.5 (89%)
P3 ($N = 89$)				
ideal delta-area (red.)	2.94e-07 (20%)	2.87e-07 (21%)	2.17e-07 (9%)	2.16e-07 (10%)
max. delta-area (red.)	5.12e-07 (12%)	5.06e-07 (13%)	4.35e-07 (5%)	4.34e-07 (5%)
CPU-time (red.)	0.33 (82%)	0.26 (86%)	1.24 (77%)	0.63 (88%)
P4 ($N = 98$)				
ideal delta-area (red.)	6.12e-07 (22%)	6.11e-07 (22%)	5.60e-07 (5%)	5.64e-07 (4%)
max. delta-area (red.)	9.10e-07 (16%)	9.09e-07 (16%)	8.65e-07 (2%)	8.69e-07 (1%)
CPU-time (red.)	0.41 (81%)	0.31 (85%)	1.73 (78%)	0.93 (88%)
P5 ($N = 225$)				
ideal delta-area (red.)	1.40e-08 (33%)	1.32e-08 (37%)	1.16e-08 (28%)	1.29e-08 (20%)
max. delta-area (red.)	4.00e-08 (15%)	2.91e-08 (38%)	2.76e-08 (14%)	2.88e-08 (10%)
CPU-time (red.)	1.12 (80%)	1.21 (79%)	1.62 (80%)	1.51 (81%)
P6 ($N = 500$)				
ideal delta-area (red.)	1.70e-06 (17%)	1.82e-06 (11%)	1.07e-06 (18%)	1.43e-06 (-9%)
max. delta-area (red.)	3.55e-06 (10%)	3.72e-06 (6%)	2.07e-06 (11%)	2.46e-06 (-6%)
CPU-time (red.)	6.98 (76%)	5.89 (80%)	12.24 (74%)	7.55 (84%)
P7 ($N = 1000$)				
ideal delta-area (red.)	6.22e-06 (11%)	6.45e-06 (8%)	4.89e-06 (12%)	4.67e-06 (16%)
max. delta-area (red.)	2.03e-05 (7%)	2.14e-05 (2%)	1.49e-05 (4%)	1.47e-05 (6%)
CPU-time (red.)	33.26 (70%)	28.11 (74%)	64.36 (69%)	43.8 (79%)

Table 6.3: Test results for the combination algorithm on problems with either 5-10-40-Constraint or maximum cardinality constraint. The 40 points in the first part were generated by an MIQP solver, with results for equidistant spacing and 2-Phase spacing of the points reported separately. The reduction of the max. delta-are, ideal delta-area, and CPU-time in comparison to just applying the MIQP solver with 400 points and 2-Phase distribution is reported in parentheses. The maximum cardinality was chosen as in all previously reported tests: $K = 4$ for P1-P4, $K = 8$ for P5. Algorithm runtime was measured in seconds.

	5-10-40-Constraint		**Max. cardinality**	
	equal distr.	**2-Phase**	**equal distr.**	**2-Phase**
P1 ($N = 31$)				
ideal del.-area (red.)	2.03e-07 (0.3%)	2.02e-07 (0.5%)	1.37e-07 (18%)	1.38e-07 (18%)
max. del.-area (red.)	1.30e-06 (0.0%)	1.30e-06 (0.1%)	2.28e-07 (12%)	2.29e-07 (11%)
CPU-time (red.)	0.91 (87%)	0.82 (88%)	0.75 (81%)	0.45 (89%)
P2 ($N = 85$)				
ideal del.-area (red.)	3.42e-08 (0.4%)	3.42e-08 (0.4%)	4.43e-07 (2.6%)	4.44e-07 (2.3%)
max. del.-area (red.)	1.19e-06 (0.0%)	1.19e-06 (0.0%)	8.12e-07 (1.4%)	8.14e-07 (1.3%)
CPU-time (red.)	5.29 (87%)	4.47 (89%)	216.4 (72%)	104.5 (86%)
P3 ($N = 89$)				
ideal del.-area (red.)	1.80e-08 (3.4%)	1.80e-08 (3.4%)	2.14e-07 (2.6%)	2.14e-07 (2.5%)
max. del.-area (red.)	4.75e-07 (0.1%)	4.75e-07 (0.1%)	4.32e-07 (1.3%)	4.32e-07 (1.2%)
CPU-time (red.)	6.17 (86%)	5.41 (88%)	583.0 (72%)	401.6 (81%)
P4 ($N = 98$)				
ideal del.-area (red.)	6.27e-08 (2.8%)	6.27e-08 (2.8%)	5.22e-07 (2.6%)	5.24e-07 (2.2%)
max. del.-area (red.)	1.24e-06 (0.1%)	1.24e-06 (0.1%)	8.05e-07 (1.7%)	8.07e-07 (1.5%)
CPU-time (red.)	8.26 (88%)	8.56 (88%)	18851 (76%)	8546 (89%)
P5 ($N = 225$)				
ideal del.-area (red.)	7.02e-08 (0.7%)	6.99e-08 (1.2%)	1.09e-08 (24%)	1.09e-08 (24%)
max. del.-area (red.)	1.63e-06 (0.0%)	1.63e-06 (0.0%)	2.68e-08 (11%)	2.68e-08 (11%)
CPU-time (red.)	18.22 (88%)	17.55 (88%)	29.4 (85%)	20.7 (89%)
P6 ($N = 500$)				
ideal del.-area (red.)	4.74e-06 (1.3%)	4.74e-06 (1.2%)	n.a.	n.a.
max. del.-area (red.)	3.11e-04 (0.0%)	3.11e-04 (0.0%)	n.a.	n.a.
CPU-time (red.)	235.4 (85%)	189.9 (88%)	n.a.	n.a.
P7 ($N = 1000$)				
ideal del.-area (red.)	8.26e-06 (2.1%)	8.27e-06 (2.0%)	n.a.	n.a.
max. del.-area (red.)	1.46e-03 (0.0%)	1.46e-03 (0.0%)	n.a.	n.a.
CPU-time (red.)	1107 (87%)	987.4 (89%)	n.a.	n.a.

On problem instances with 5-10-40-Constraint, irrespective of choice of placement scheme and heuristics, the improvement of the solution quality is quite significant (up to 69%). Additionally, the algorithm runtime is far shorter than if just the "normal" ϵ-Constraint methods are used. The 2-Phase distribution scheme does seem to perform equally well or slightly better than the equidistant placement for both of the two heuristics. For the two largest problems, the runtime advantage of the combination algorithm is not as

pronounced, but never falls below 25%. The main cause for this is probably that a larger number of envelopes had to be calculated, and each of those has more corner portfolios than the envelopes of the smaller problem instances, and a single step to calculate the next corner portfolio is quite costly due to the large covariance matrices. The accumulation of these three effects probably swallows up much of the time we saved in the first part.

Perhaps most surprising are the results of the 2-solver-calls heuristic. Although it has been mainly developed to quickly calculate a feasible portfolio of acceptable quality, when it is combined with the envelope approach it delivers solutions that are nearly on par with the n-solver-calls heuristic – and their calculation requires less time.

The reason for this unexpected success becomes clear when we look at Figure 6.1 which depicts the solution of the 2-solver-calls heuristic (40 point budget) and the resulting envelopes for Problem 1: the original portfolios from the 2-solver-calls heuristic are completely dominated by the envelopes. The explanation for this is to be found in the processing step that determines how the sets of convex constraints are constructed from the calculated points. For every solution of the point-based heuristic we don't just copy the set of convex constraints from the ϵ-Constraint problem and calculate the respective envelope, but instead we generate a different set of convex constraints. This new set is based on the shares that have an actual weight of larger than 5% in the solution, and only those assets are included in the 40%-Constraint, which results in the constraint to be "less binding".

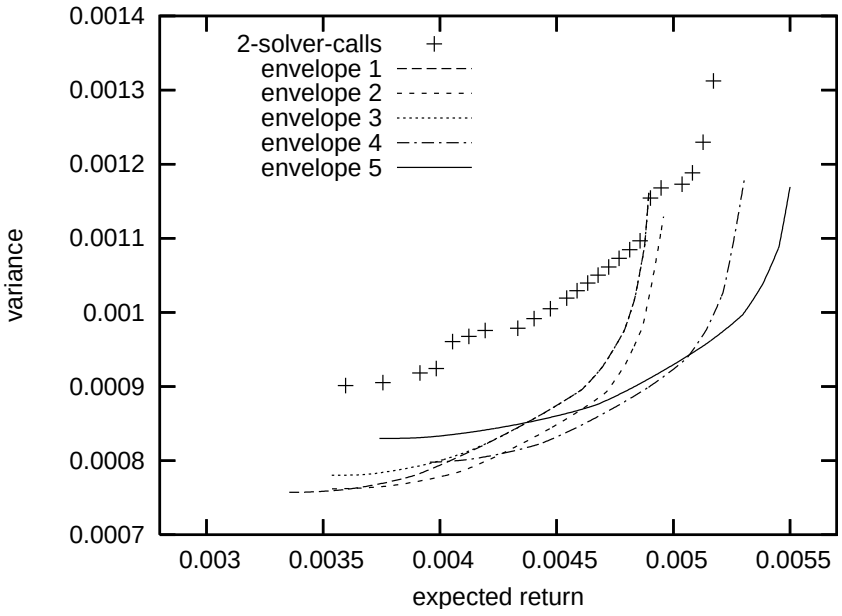


Figure 6.1: Nondominated portfolios calculated with the 2-solver-calls heuristic (40 point budget) and the resulting envelopes generated with the information from these portfolios. Problem instance: P1 (31 assets) with 5-10-40-Constraint.

Figure 6.2 gives an impression of the disparities in solution quality for the simple point-based algorithms and their combination with the envelope-based approach. Even though the n-solver-calls heuristic is considerably closer to the results of the combination algorithms, the difference is still quite obvious. The results for the two combination algorithms, however, are nearly identical. The main difference is that the combination with n-solver-calls heuristic was able, for this problem instance, to get solutions slightly beyond the ends of the front generated by the 2-solver-calls-combination.

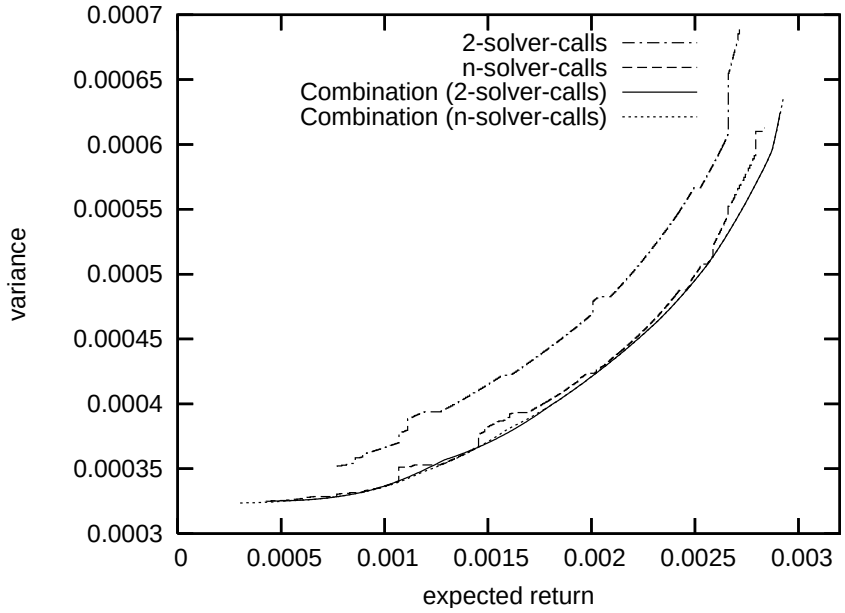


Figure 6.2: Solution of the 2-solver-calls heuristic (400 point budget), the n-solver-calls heuristic (400 point budget), the combination algorithm with 2-solver-calls heuristic (40 point budget), and combination algorithm with n-solver-calls heuristic (40 point budget). For all point-based heuristics (both standalone and as part of the combination algorithm), the 2-Phase distribution scheme was used. Problem instance: P5 (225 assets) with 5-10-40-Constraint.

When either the reoptimization heuristic or the n-solver-calls heuristic is combined with the envelope approach to solve problems with a **maximum cardinality constraint** (see Table 6.2), the runtime reduction is consistently above 70%, and no relative increase in runtime can be detected for larger problem sizes. The simple explanation for this is that the calculation of the envelopes for, e.g., P7 does on average not require more time than that of P5, as we eliminate all shares that are set to zero. Thus the covariance matrix is shrunk to size $K \times K$ before the parametric quadratic programming solver is applied. The improvements to the solution quality for the heuristics in the maximum cardinality case are not as high as the ones in the 5-10-40-Constraint case. This is probably a consequence of the fact that the solution quality of the heuristics in cardinality con-

strained problems was by far better to begin with. Nevertheless, in nearly all scenarios, an improvement is still noticeable.
Based on the results for the seven problems, it is difficult to come to a conclusive answer whether the equidistant distribution or the 2-Phase distribution scheme is superior, as there are several scenarios in which either of the two yields better results.
When we compare the two heuristics, the n-solver-calls heuristic in most cases justifies its longer computation time by generating better results. The simpler reoptimization heuristic is not far off, but this is not as surprising as above, since even for the standard point-based approach (cf. Chapter 4), the results of this heuristic were not much worse than those of the n-solver-calls heuristic.

The combination of an MIQP solver with the envelope approach (Table 6.3), has only a very limited potential for solution improvement, as all the points calculated by the MIQP solver are guaranteed to be optimal. The only improvement is to be found in the ability of the combination algorithm to change the stair-shaped nondominated front (cf. Figure 4.3) into a front consisting of envelopes (cf. Figure 4.4). This effect can be observed in the improvement of the ideal delta-area. As the right edge (point with maximized expected return) and left edge (MVP) of the front is determined by the MIQP solver, irrespective of whether the point-based approach or the combination algorithm is used, the additional areas that are part of the max. delta-area are identical. Therefore the relative reduction of the max. delta-area is much smaller.
Due to the longer time it takes to calculate a single point with the MIQP solver, the additional cost for computing the envelopes and the aggregated front does not matter as much as in case of the heuristics. The runtime improvement is therefore close to the possible maximum of 90% for all problem instances, irrespective of the distribution scheme and the nonconvex constraint type.
Again, a reliable statement whether the equidistant or the 2-Phase distribution is better can not be made. The results are quite close for all problems and both problem types. The advantage of the 2-Phase procedure to distribute the points in a way that not many points are wasted beyond the left end of the Pareto front does not come into play here, as the left end can be – and is – determined exactly with the MIQP solver. The simpler equidistant approach does therefore seem to be sufficient here.

In order to give the reader a better impression how the different algorithms presented in the thesis compare with respect to both computation time and solution quality, we have conducted a series of additional tests on Problem 5 with 5-10-40-Constraint. In order to construct "simulated" convergence curves, we varied the point budget (and thus also the computation time) for the "pure" ϵ-Constraint approaches and their combination counterparts. The results are documented in Figure 6.3. For the 2-solver-calls heuristic, the n-solver-calls heuristic, and the combinations of envelope-based approach and these heuristics, the point budget was set to values between 20 and 1000, for the MIQP-solver and its envelope-based extension, we started with 10 points and raised the budget to 700. For the P-MOEA and the E-MOEA, "real" convergence curves had been already calculated in Chapter 5. Due to the fact that the clock speed of the computer on which we performed the tests for the E-MOEA and the P-MOEA was slightly higher (1.6 GHz)

than the one on which all the other tests were done (1.5 GHz), the measured computation times had to be adjusted to compensate for this.
As we tried to avoid the effect that the results are mainly influenced by the two portfolios with either the lowest variance or the highest return, Figure 6.3 uses the ideal delta-area to measure solution quality.

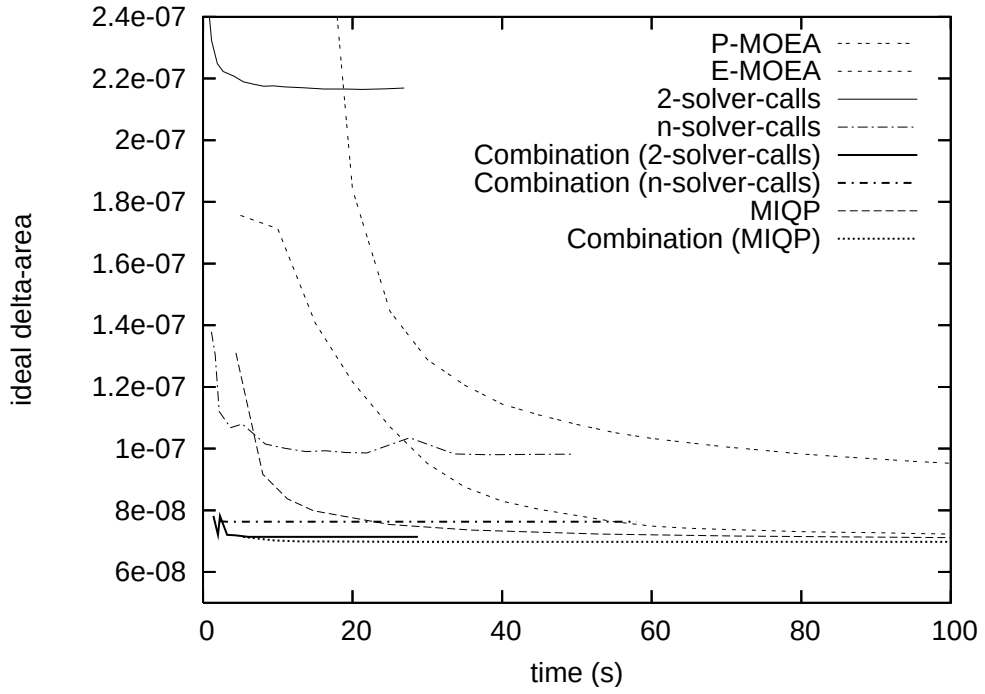


Figure 6.3: Convergence curves of the 2-solver-calls heuristic, the n-solver-calls heuristic, the P-MOEA and the E-MOEA, the MIQP solver and three combination algorithms (envelope-based with 2-solver-calls, n-solver-calls and MIQP) for P5 (225 assets) with 5-10-40-Constraint.

We have to emphasize that there is an important distinction between the inner workings of the P-MOEA and the E-MOEA on the one side and all the other algorithms on the other sides. This difference is critical to the interpretation of Figure 6.3: The convergence curves of the P-MOEA and the E-MOEA depict the average result of a number of algorithm runs over time. The "simulated" curves of the other algorithms have been constructed by applying the respective procedure repeatedly for the different budgets. Due to the structure of the ϵ-Constraint algorithms, there is no easy way to calculate e.g. a solution with a 300 point budget by just inserting an additional 100 points into the solution that we got with a 200 point budget. Distributing the points across the expected return axis would be easy to realize, but the solution thus calculated would not be the same as the one we got when we started with 300 points, and its quality would probably be worse.

Evolutionary algorithms are able to provide a solution regardless how much time they are given, and, given more time, the solution can be improved without having to restart the algorithm. This is certainly an advantage, and the other algorithms presented here would require extensive redesign to gain the same capability.

Given the information of the previous tests, the results in Figure 6.3 are, for the most part, as expected. The P-MOEA converges quite slowly, the E-MOEA is faster and – if given enough time – gets very close to the optimal solution. The 2-solver-calls heuristic is able to provide a solution in a very short amount of time, but the quality is not overwhelming, and even with a large point budget, it does not improve much. After little more than 20 seconds, all the other algorithms are superior. The results of the n-solver-calls heuristic are better, but the heuristic suffers from the same basic problem as the 2-solver-calls heuristic, namely its results cannot be improved significantly beyond a certain level, which is still a good distance away from the actual optimum, even if a large number of points is used for the approximation [4].

The results for the MIQP solver are quite good, and if there is enough time to calculate a larger number of points, the approximation will be nearly perfect. The only algorithm clearly superior is the combination of the MIQP solver with the envelope approach. After 10 seconds at most it provides results that are better than all the results the other heuristics are able to calculate, even when they are given much more time.

Should no MIQP solver be available, or would the calculation of even very few points take longer than we are willing to wait and we need to get a quick solution, the "extension" of the 2-solver-calls heuristic or the n-solver-calls heuristic with the envelope approach are the best choice. Curiously, the combination algorithm that is based on the 2-solver-calls heuristic is able to compute a better approximation than the one with the n-solver-calls heuristic for nearly all tested point budgets. This does seem to be a problem specific effect. But the results for both combination algorithms are nevertheless quite remarkable, especially when compared to the results of their point-based counterparts.

6.3 Summary: Combination of Point-Based and Envelope-Based Approaches

In this Chapter, we have presented an algorithm that uses the different ϵ-Constraint approaches from Chapter 4 to determine sets of convex constraints that define envelopes of feasible – and hopefully good – solutions. These envelopes are then calculated and accumulated to an aggregated Front.

Based on extensive tests, we were able to show that this algorithm delivers results that are significantly better in most scenarios. The combination algorithm was able to calculate these often superior solutions in a fraction of the time expended by the "simple" point-based procedures. It could do this since the information from a single point is enough to determine an envelope, which in turn defines a complete set of feasible

[4] Therefore the convergence curves of the ϵ-Constraint heuristics have not been calculated for budgets of more than 1000 points.

solutions. A relatively small number of points can therefore represent enough information to effectively describe the whole Pareto front.
Every type of point-based solution can be post-processed this way, irrespective of the algorithm guaranteeing optimality for each point or not. Even portfolios that have been constructed using a point-based evolutionary algorithm (cf. Section 5.3) could be treated in this fashion, and thus their solution could be improved without too much effort. An envelope-based post-processing might be especially worthwhile if only a small number of points on or near the Pareto front is known.
One main advantage of the combination algorithm is a description of the Pareto front that no longer is made up of just a collection of points, but instead consists of a set of curves (parabola segments) that define a solution for the whole breadth of the aggregated front.
The algorithm we presented was just a simple but very effective realization of the principle to combine point-based and envelope-based approaches. All of the more complex building blocks of the procedure – (i) the construction of a feasible solution given a minimum expected return, (ii) the calculation of the envelopes, and (iii) their fusion into an aggregated front – were already available as part of the algorithms presented in the previous chapters[5]. The only drawback of the algorithm, in our opinion, is the limited availability of these building blocks, especially of the parametric quadratic programming algorithm and the algorithm that calculates the aggregated front, as there are no publicly available libraries that are efficient and offer this functionality. If all the parts have to be implemented from scratch, the required effort is quite extensive.
Neither the point-based heuristic nor the distribution schemes we used were especially designed towards this type of algorithm. Therefore, there is a good chance that other methods to calculate feasible points may be even more effective in combination with the envelope approach. This is a topic we intend to cover in more detail in the future.

[5]The only task not yet realized was the analysis of the points to determine the sets of convex constraints, which was – in comparison to the other algorithm parts – easy to implement.

7 Summary and Conclusion

The goal of this thesis was to develop, document, and test different approaches that are able to efficiently calculate a sufficiently precise approximation of the complete Pareto front for mean-variance portfolio selection problems with nonconvex constraints.
The main content of this thesis can be summarized as follows:
After a brief introduction in Chapter 1, the economic principles behind the mean-variance model have been presented in Chapter 2, together with the model itself and a short description of the methodology of the three main solution approaches: the weighting method, the ϵ-Constraint approach, and parametric quadratic programming. We have also briefly touched upon other risk measures besides the variance of the portfolio. Additionally, the different types of constraints for portfolio selection have been listed in Chapter 2 and we have analyzed them regarding the possibility to easily integrate them into the standard mean-variance optimization model. The constraints for which this is not possible – the so called *nonconvex constraints* – are the basis of the problems we have addressed in Chapters 4, 5, and 6. The three constraints of this type we have focused on are (i) the 5-10-40-Constraint based on §60(1) of the German investment law, (ii) maximum cardinality constraint, and (iii) buy-in thresholds.
Before tackling problems with nonconvex constraints, however, an efficient and numerically stable parametric quadratic programming (PQP) algorithm for "standard" mean-variance optimization has been presented in Chapter 3. This algorithm is an important building block for techniques used in later chapters. Due to several algorithmic improvements, our PQP solver is capable of calculating the complete Pareto front in a very short amount of time even on large problem instances. We have also provided some advice regarding the implementation of the algorithm by presenting two matrix representations and by demonstrating how they can be used in different parts of the PQP algorithm to further enhance efficiency.
Chapter 4 describes the ϵ-Constraint approach to generate a point-based approximation for portfolio selection problems with any one of the three nonconvex constraints. Two main aspects of the ϵ-Constraint method have been discussed in detail: how to distribute a given point budget along the Pareto front as efficiently as possible, and how to get good results for each of those points. Based on a series of tests conducted on a set of benchmark problems, we demonstrated that the 2-Phase algorithm we developed for the efficient distribution of points is able to calculate results of significantly better quality than we would get by spacing points equidistantly on the expected return axis. We also proposed several heuristics designed especially for the calculation of each point. As expected, the more time-consuming heuristics are, in most cases, able to produce superior results. By transforming the basic mean-variance model into a mixed-integer model, we were able to replace the heuristics with a commercial solver that guarantees

solution optimality for each point. The results thus gained are as good as a point-based approximation can be, but the time necessary to calculate them can be prohibitively long – especially for problems with a maximum cardinality constraint.
In Chapter 5, we have presented a new algorithm, the envelope-based multi-objective evolutionary algorithm (E-MOEA), which is a combination of a multi-objective evolutionary algorithm and an embedded PQP solver. The task of the MOEA is to find convex subsets of the search space. Then, for each subset the PQP algorithm calculates the Pareto front, which we call an *envelope*. These envelopes are then combined to form the overall solution to the problem. For comparison, we also implemented and tested a state-of-the-art multicriteria evolutionary algorithm (P-MOEA) by Streichert et al. [SUZ03, SUZ04a, SUZ04b]. This algorithm had been designed specifically for portfolio selection problems with nonconvex constraints. The test results of our E-MOEA are much better than those of the P-MOEA with respect to convergence rate, solution quality during the algorithm run, and final solution quality.
And finally, in Chapter 6, we have proposed to combine the envelope-based approach of the E-MOEA with the ϵ-Constraint algorithms from Chapter 4. The intention was to find the most promising convex subsets with the aid of "good" points that have already been calculated by the point-based algorithms. Therefore, after computing a certain number of points, they are analyzed and the convex subsets they belong to are identified. Then, the envelopes that are associated with the subsets are determined and, in a last step, the aggregated front is computed. For the first part of the combination algorithm – the calculation of a set of points on or near the Pareto front – ϵ-Constraint heuristics or an MIQP solver can be used. Tests have shown that a much smaller number of points than we used for a point-based approximation is usually enough to determine most of the relevant envelopes. Since the calculation of the respective number of envelopes is not very time consuming, the overall computation times of the combination algorithms are significantly shorter than those of the "pure" ϵ-Constraint methods, and the solution quality is improved as well, especially when we combine the envelope approach with the faster heuristics.
Based on our experiences and the results of the previous chapters, we provide a brief summary of the strengths and weaknesses of the presented algorithms:

1. **P-MOEA:**
   The point-based multi-objective evolutionary algorithm (P-MOEA) is easy to implement and very flexible, since the variance can be replaced by other dispersion measures like e.g. CVaR or VaR, and there are several different ways to integrate constraints. Generated solutions are of poor quality especially on large problem instances, and their calculation requires a comparably long time.

2. **E-MOEA:**
   The envelope-based multi-objective evolutionary algorithm (E-MOEA) is more complicated, due to two nontrivial core elements, the PQP solver and the aggregation of the envelopes. The results are better than those of the P-MOEA, and the algorithm converges faster. It is especially well suited for problems with maximum cardinality constraint, since for those, it delivers nearly perfect results

in comparatively short time. Its results are not point-based, i.e. the output of an algorithm run is a functional description of the nondominated frontier that allows to easily calculate all portfolios on the front. This is a significant improvement over point-based approaches which produce just a collection of points.

3. **Point-based ϵ-Constraint heuristics (2-solver-calls/reoptimization and n-solver-calls):**
   The heuristics are able to calculate feasible solutions very fast, but often of minor quality. If given enough time, even the P-MOEA might be able to produce a better approximation.

   The n-solver-calls heuristic calculates fewer points in the same time, but its results are normally much better than those of the 2-solver-calls heuristic, at least for problem instances with 5-10-40-Constraint. Both heuristics are easy to implement if a fast and reliable external QP solver is available. The choice of solver has a huge influence on the speed of the heuristics, since the accumulated time for all the solver calls represents the bulk of overall algorithm runtime.

4. **MIQP solver:**
   If a fast mixed-integer solver is available to calculate the single points, the solution it produces is nearly perfect, since every point is on the Pareto front. The main drawbacks of this approach are the long time it requires to calculate a solution and that this solution is point-based. Its results are therefore only an approximation of the Pareto front even though all the calculated points are situated on the front. A main problem for the implementation is that there are only very few libraries with algorithms that have the capability and speed to solve mixed-integer quadratic programming problems in an acceptable time, and – to our knowledge – they are all commercial and the licenses are not cheap. (With a library that contains such an algorithm, however, the implementation can be done fairly easily.)

5. **Combination algorithms:**
   These algorithms are much faster than the point-based algorithms (2-solver-calls, n-solver-calls, or the MIQP solver) they are based on, and their results are superior as well. Like the solution computed by the E-MOEA, the result of every combination algorithm is not just a collection of points but an envelope-based description of the nondominated frontier. The main drawback is that the algorithm is more complex and requires significantly more effort to implement than the purely point-based solution approaches. But once a working implementation is available, there is no good reason to still use the purely point-based approaches.

Which of the algorithms presented in this thesis is actually the best strongly depends on the preferences of the person that has to use it – and also on the availability of those algorithm components that cannot be implemented in short order, like e.g. a high-performance MIQP solver. The description and analysis of the different approaches we have provided hopefully help the reader in selecting the one that is most suited to his needs.

We will conclude this chapter and the whole thesis with a brief outlook on what we think would be the most promising extensions of the presented algorithms, and what future research aspects remain to be tackled.
One idea for an algorithm modification that is easy to realize and might prove interesting is the combination of the point-based heuristics and the envelope-based MOEA. The simpler heuristics could be used to "jump-start" the E-MOEA by providing an initial starting solution of fairly good quality. Ideally, such an algorithm would nevertheless converge to a very good solution, but this result could be achieved in much less time than when the E-MOEA were initialized randomly. Compared to the algorithms presented in Chapter 6, this new combination might have the additional advantage that the solution quality could be improved by just extending the runtime of the algorithm.
We think, however, that the most promising extension of the techniques presented here could be the integration of both variable and fixed transaction costs into the mean-variance framework: It is not too difficult to modify the parametric quadratic programming algorithm from Chapter 3 in a way that enables it to cope with variable transaction costs. Additionally, fixed transaction costs can probably be addressed by introducing binary variables in a similar fashion as was the case with e.g. cardinality constraints. With the integration of those techniques and the respective modifications to the "non-convex" algorithms we think there is a good chance that we would not only be able to model the problems with both fixed and variable transaction costs, but to provide their solution in an acceptable amount of time. This could be of great practical relevance when managing real-world portfolios.
The sensitivity of the results with respect to estimation errors in the input data is an aspect we have deliberately ignored in this thesis. Several publications have dealt with this in a standard mean-variance setting, but none of them, at least as far as we know, has included nonconvex constraints. It is not too difficult to construct a worst case scenario in which tiny changes in the input data result in completely different solutions in the portfolio space, but we think that only extensive tests based on real world data and commonly used estimation techniques could provide meaningful results.
Finally, extending the presented algorithms to a multiperiod or continuous setting would be particularly challenging, since, among many other difficulties, this would mean that compliance with all constraints probably has not only to be guaranteed at the beginning of the single period, but at the beginning of each period (multiperiod model) or even during the whole investment period (continuous model).

A Further Test Results

A.1 Heuristics for Problems with 5-10-40-Constraint

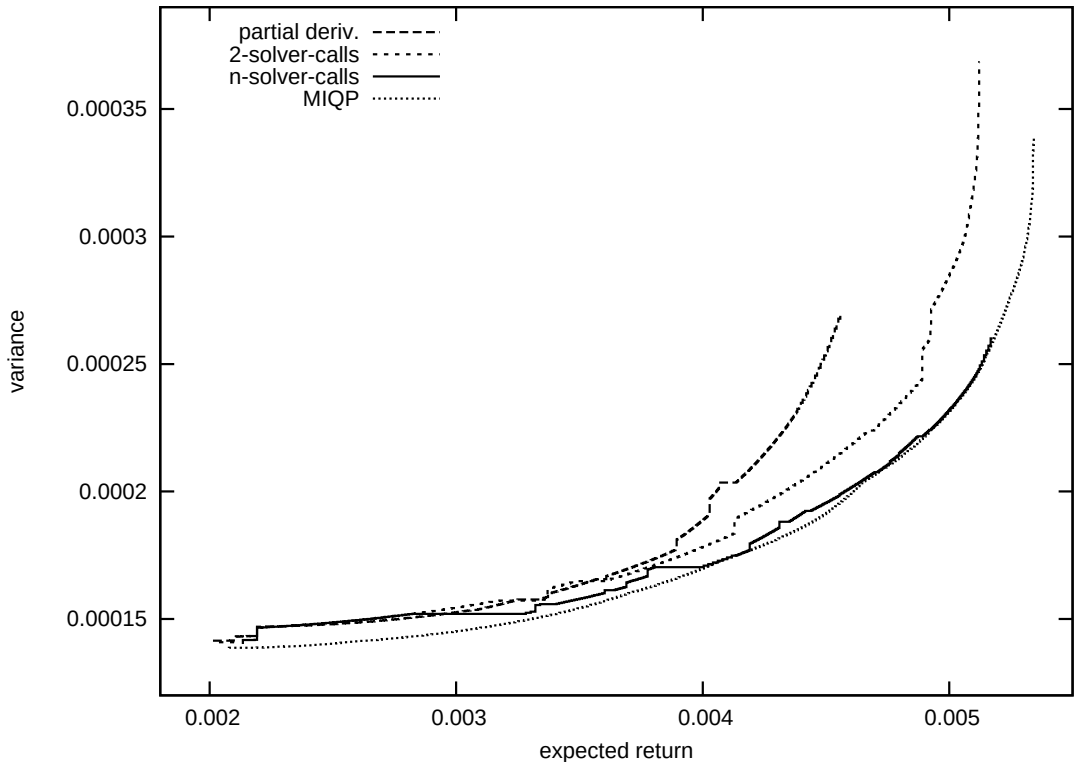


Figure A.1: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for Problem 2 (85 assets) with 5-10-40-Constraint

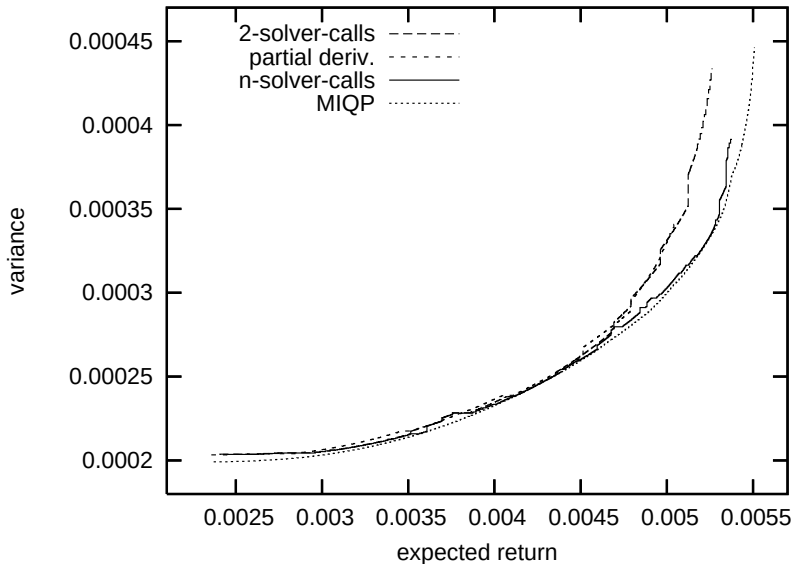


Figure A.2: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for Problem 3 (89 asset universe) with 5-10-40-Constraint

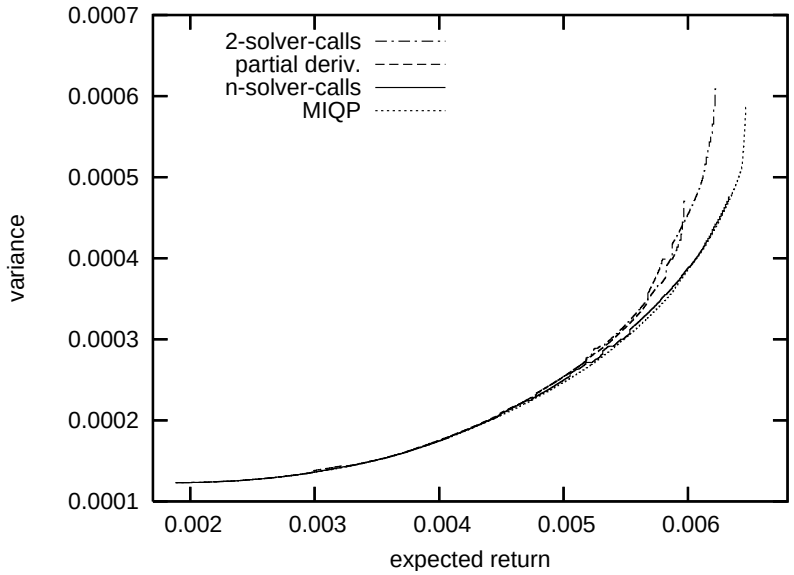


Figure A.3: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for Problem 4 (98 asset universe) with 5-10-40-Constraint

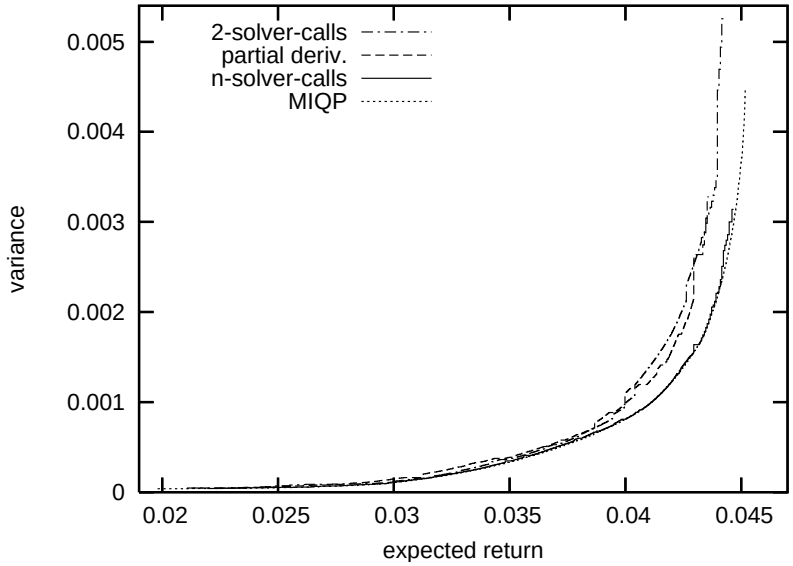


Figure A.4: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for Problem 6 (500 asset universe) with 5-10-40-Constraint

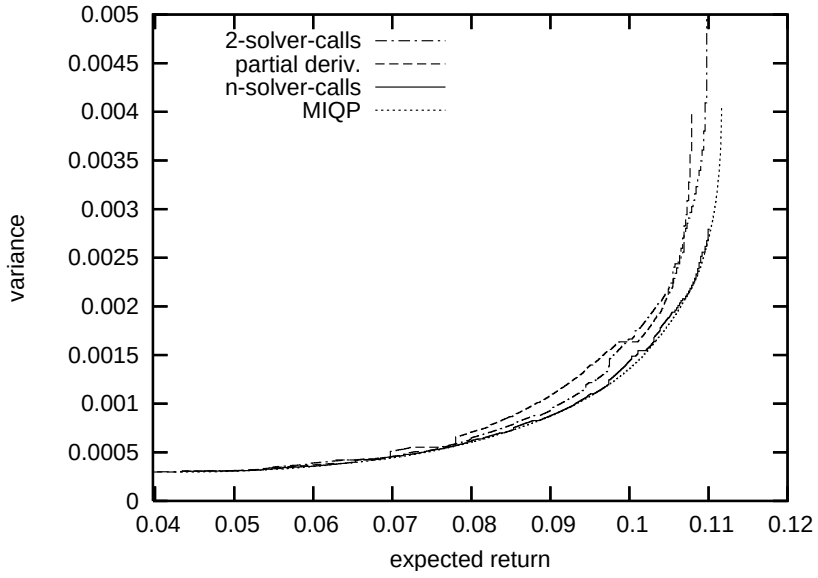


Figure A.5: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, the partial derivatives heuristic, and the MIQP solver for Problem 7 (1000 asset universe) with 5-10-40-Constraint

A.2 Heuristics for Problems with a Maximum Cardinality Constraint

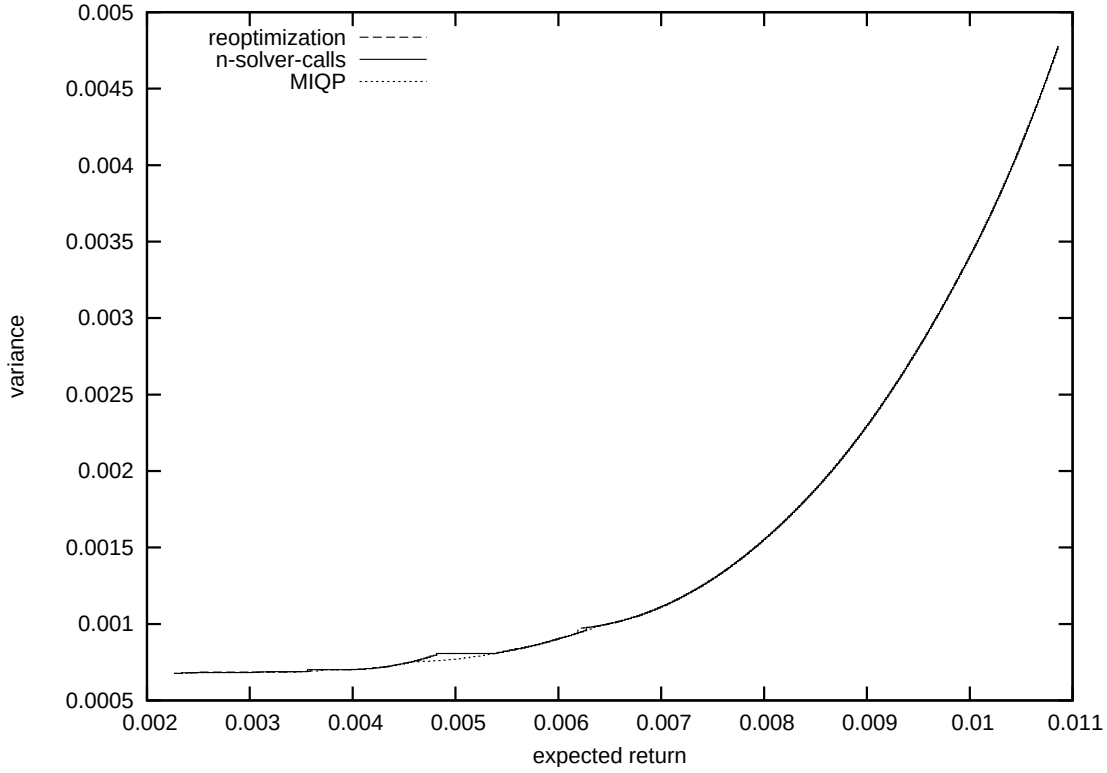


Figure A.6: Results of the reoptimization heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 1 (31 assets) with a maximum cardinality constraint of $K = 4$

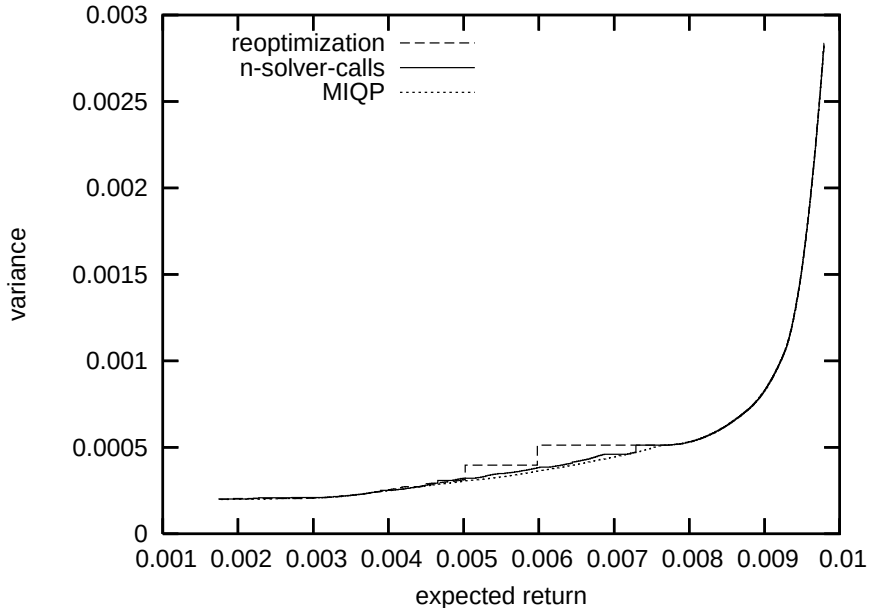


Figure A.7: Results of the reoptimization heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 2 (85 assets) with a maximum cardinality constraint of $K = 4$

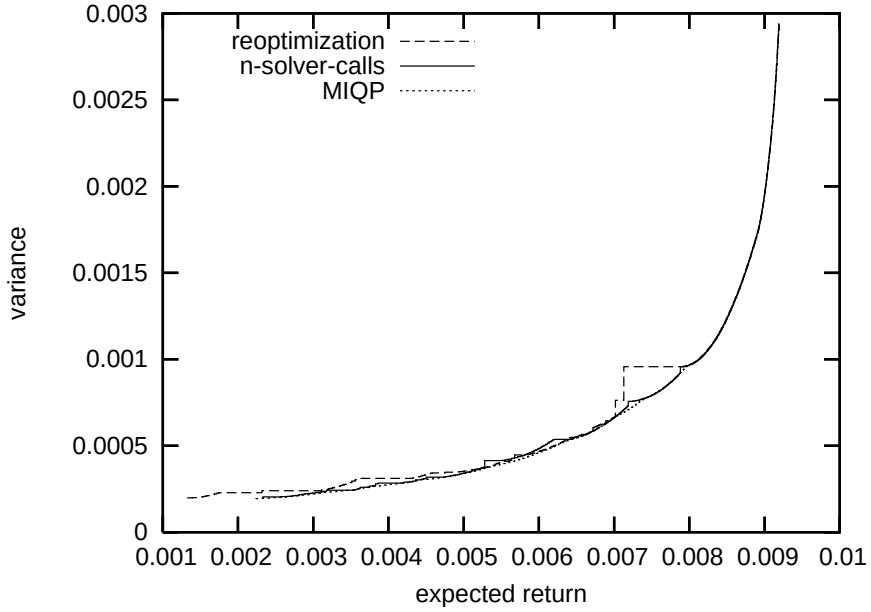


Figure A.8: Results of the reoptimization heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 4 (98 assets) with a maximum cardinality constraint of $K = 4$

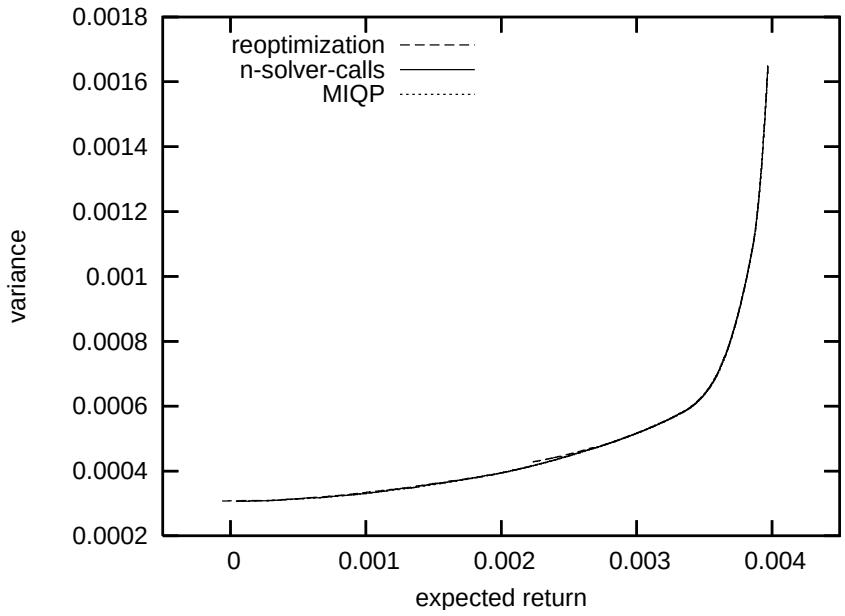


Figure A.9: The ideal front and the results of the reoptimization heuristic and the n-solver-calls heuristic for Problem 5 (225 assets) with maximum cardinality constraint of $K = 8$.

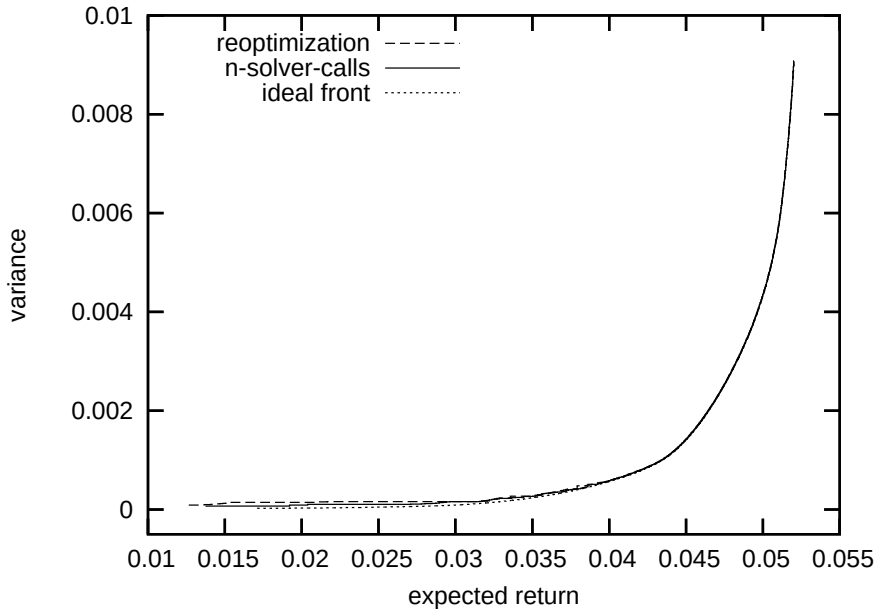


Figure A.10: The ideal front and the results of the reoptimization heuristic and the n-solver-calls heuristic for Problem 6 (500 assets) with maximum cardinality constraint of $K = 8$.

A.3 Heuristics for Problems with Buy-In Thresholds

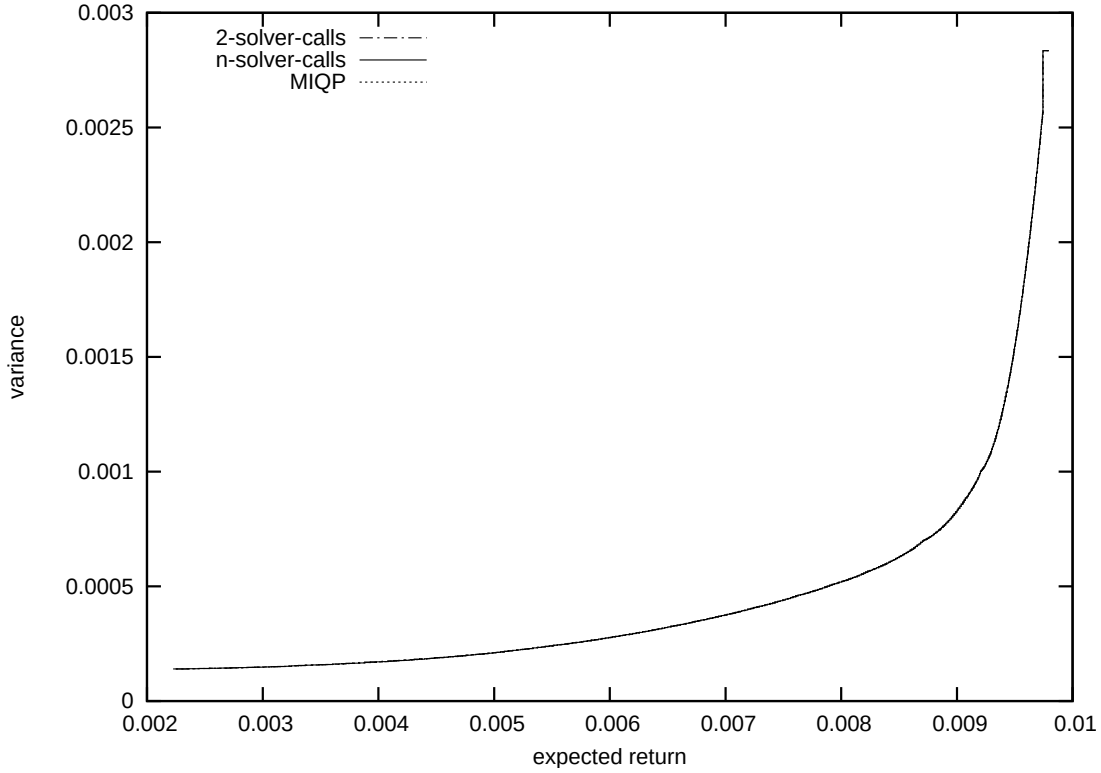


Figure A.11: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 2 (85 assets) with a buy-in threshold of 0.05

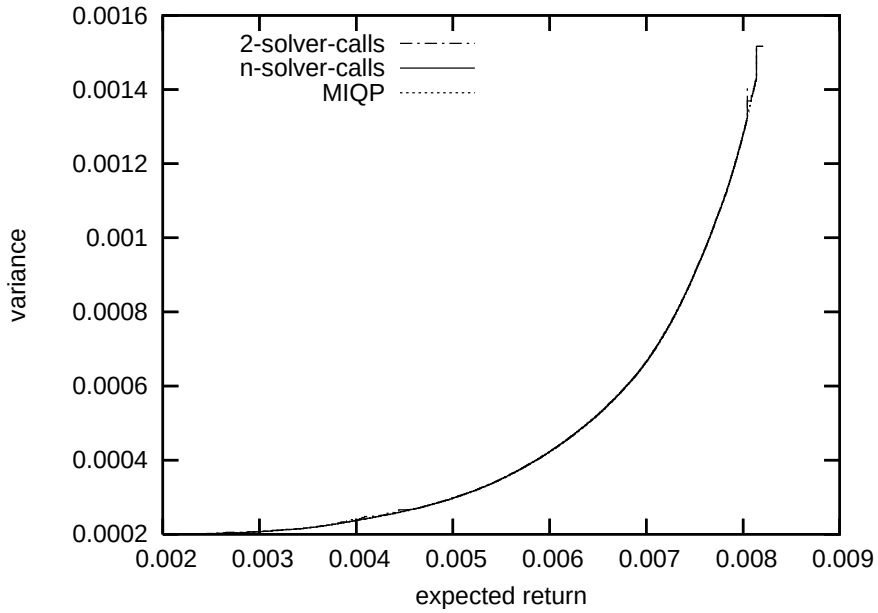


Figure A.12: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 3 (89 assets) with a buy-in threshold of 0.05

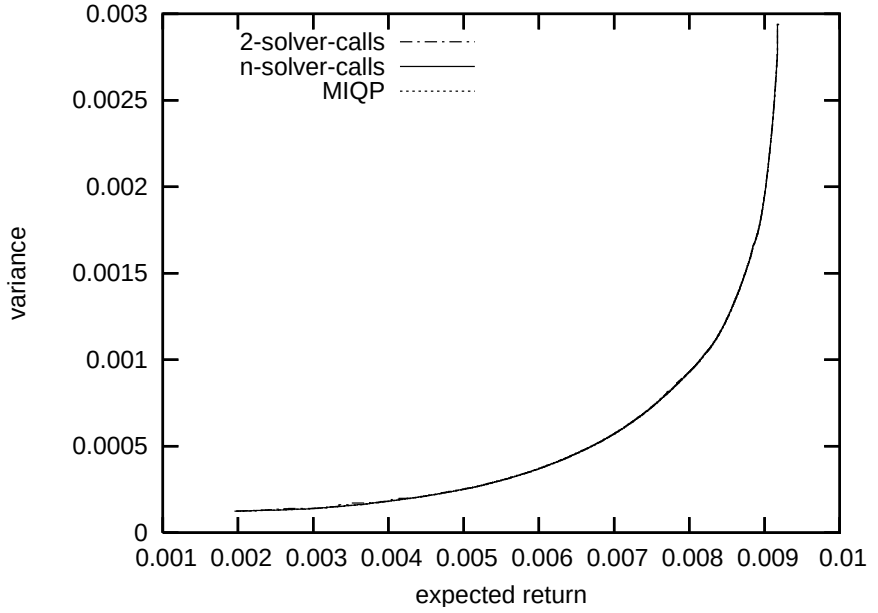


Figure A.13: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 4 (98 assets) with a buy-in threshold of 0.05

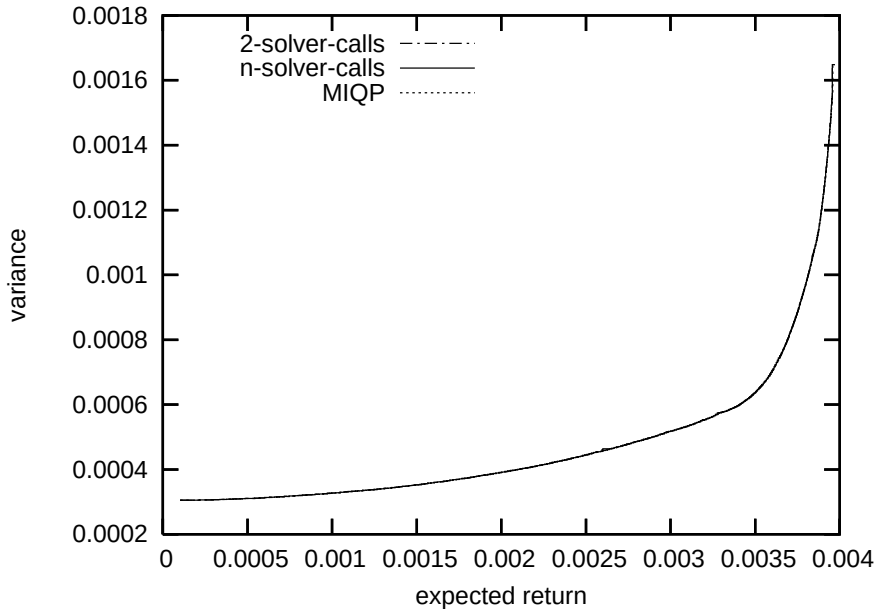


Figure A.14: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 5 (225 assets) with a buy-in threshold of 0.05

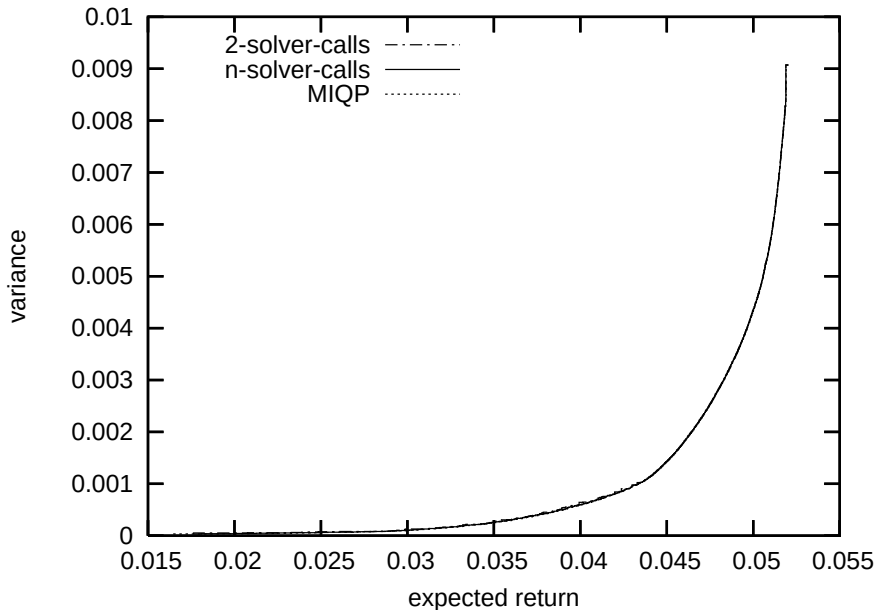


Figure A.15: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 6 (500 assets) with a buy-in threshold of 0.05

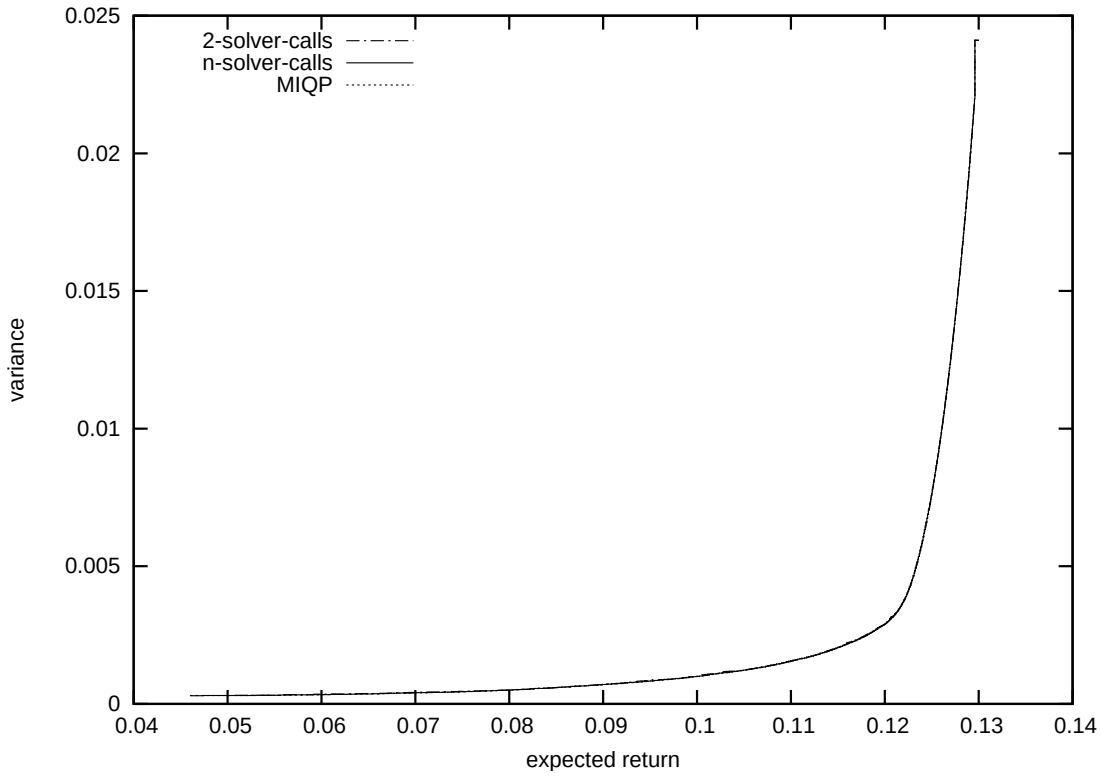


Figure A.16: Results of the 2-solver-calls heuristic, the n-solver-calls heuristic, and the MIQP solver for Problem 7 (1000 assets) with a buy-in threshold of 0.05

References

[AB04] G. J. Alexander and A. M. Baptista. A comparison of VaR and CVaR constraints on portfolio selection with the mean-variance model. *Management Science*, 50(9):1261–1273, 2004.

[ADEH99] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[AL05] R. Armananzas and J. A. Lozano. A multiobjective approach to the portfolio optimization problem. In *Congress on Evolutionary Computation*, pages 1388–1395. IEEE, 2005.

[ATL06] Automatically Tuned Linear Algebra Software (ATLAS). online: `http://math-atlas.sourceforge.net/`, 2006. Last accessed: 2006-10-18.

[Bea06] J. E. Beasley. OR-library. online, `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, 2006. Last accessed: 2006-10-18.

[Bes96] M. J. Best. An Algorithm for the Solution of the Parametric Quadratic Programming Problem. In H. Fischer, B. Riedmüller, and S. Schäffler, editors, *Applied Mathematics and Parallel Computing*, pages 57–76. Physica, 1996.

[BFG+00] R. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mip: Theory and practice – closing the gap. In M. J. D. Powell and S. Scholtes, editors, *System Modelling and Optimization: Methods, Theory, and Applications*, pages 19–49. Kluwer Academic Publishers, 2000.

[BG91] M. J. Best and R. R. Grauer. Sensitivity analysis for mean-variance portfolio problems. *Management Science*, 37(8):980–989, 1991.

[BH05] M. J. Best and J. Hlouskova. An algorithm for portfolio optimization with transaction costs. *Management Science*, 51(11):1676–1688, 2005.

[Bie96] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140, 1996.

[BK00] M. J. Best and J. K. Kale. Quadratic programming for large-scale portfolio optimization. In Jessica Keyes, editor, *Financial Services Information Systems*, pages 513–529. CRC Press LLC, 2000.

[BL92] F. Black and R. Litterman. Global portfolio optimization. *Financial Analysts Journal*, 48(5):23–43, 1992.

[BLA06] Basic Linear Algebra Subprograms (BLAS). online: `http://www.netlib.org/blas`, 2006. Last accessed: 2006-10-18.

[BSS+06] J. Branke, B. Scheckenbach, M. Stein, K. Deb, and H. Schmeck. Portfolio optimization with an envelope-based multi-objective evolutionary algorithm. *European Journal of Operational Research*, submitted 2006.

[CH83] V. Changkong and Y. Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science Publishing Co., 1983.

[Chr06] C. H. Christensen. GSL--. online: `http://cholm.home.cern.ch/cholm/misc/\#gslmm`, 2006. Last accessed: 2006-10-18.

[CHT93] V. K. Chopra, C. R. Hensel, and A. L. Turner. Massaging mean-variance inputs: Returns from alternative global investment strategies in the 1980s. *Management Science*, 39(7):845–855, 1993.

[CKP03] J.-H. Cremers, M. Kritzman, and S. Page. Portfolio formation with higher moments and plausible utility. Financial economics 272-12, Revere Street Working Paper Series, 2003.

[CMBS00] T.-J. Chang, N. Meade, J. B. Beasley, and Y. Sharaiha. Heuristics for cardinality constrained portfolio optimisation. *Computers & Operations Research*, 27:1271–1302, 2000.

[CPL06] ILOG CPLEX. online: `http://www.ilog.com/products/cplex/`, 2006. Last accessed: 2006-10-18.

[CS03] Y. Crama and M. Schyns. Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research*, 150(3):546–571, 2003.

[CUs03] User's Manual CPLEX. ILOG CPLEX, Version 9.0, 2003.

[CVL02] C. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for solving multi-objective problems*. Kluwer, 2002.

[CZ93] V. K. Chopra and W. T. Ziemba. The effect of errors in means, variances, and covariances on optimal portfolio choice. *The Journal of Portfolio Management*, 19:6–11, WINTER 1993.

[DD98] I. Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[Deb01] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

[DN01] U. Derigs and N.-H. Nickel. On a metaheuristic-based DSS for portfolio optimization and managing investment guideleines. In *Metaheuristics International Conference*, 2001.

[DN03] U. Derigs and N.-H. Nickel. Meta-heuristic based decision support for portfolio optimization with a case study on tracking error minimization in passive portfolio management. *OR Spectrum*, 25:345–378, 2003.

[Edd96] D. Eddelbüttel. A hybrid genetic algorithm for passive management. Computing in economics and finance, Society of Computational Economics, 1996.

[EGBG03] E. Elton, M. Gruber, S. Brown, and W. Goetzmann. *Modern Portfolio Theory and Investment Analysis.* John Wiley and Sons, 6th edition, 2003.

[EGS06] E. Elton, M. Gruber, and J. Spitzer. Improved estimates of correlation coefficients and their impact on optimum portfolios. *European Financial Management*, 12(3):303–318, 2006.

[EKS04] M. Ehrgott, K. Klamroth, and C. Schwehm. An MCDM approach to portfolio optimization. *European Journal of Operational Research*, 155:752–770, 2004.

[Ell61] D. Ellsberg. Risk, ambiguity, and the savage axioms. *Quarterly Journal of Economics*, 75(4):643–669, 1961.

[ES03] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing.* Springer, 2003.

[Fam65] E. F. Fama. The behavior of stock-market prices. *The Journal of Business*, 38(1):34–105, 1965.

[FMP04] J. E. Fieldsend, J. Matatko, and M. Peng. Cardinality constrained portfolio optimisation. In Z. R. Yang, R. M. Everson, and H. Yin, editors, *Intelligent Data Engineering and Automated Learning*, volume 3177 of *LNCS*, pages 788–793. Springer, 2004.

[GCC06] Gnu Compiler Collection. online: `http://gcc.gnu.org/`, 2006. Last accessed: 2006-10-18.

[GH99] H. Grootveld and W. Hallerbach. Variance versus downside risk: Is there really that much difference? *European Journal of Operational Research*, 114:304–319, 1999.

[GMSW89] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45(3):437–474, 1989.

[GMW91] P. E. Gill, W. Murray, and M. H. Wright. *Numerical Linear Algebra and Optimization.* Addison-Wesley, 1991.

[Gou91] N. I. M. Gould. An algorithm for large-scale quadratic programming. *IMA Journal of Numerical Analysis*, 11:299–324, 1991.

[GP05] A. A. Gaivoronskiy and G. Pflug. Value-at-risk in portfolio optimization: properties and computational approach. *Journal of Risk*, 7(2):1–31, 2005.

[GT01] N. Gould and P. Toint. A quadratic programming bibliography. online: `www.optimization-online.org/DB_HTML/2001/02/285.html`, 2001.

[GvL96] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[HJ98] M. P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical report, Institute for Mathematical Modelling, Technical University of Denmark, Lyngby, 1998.

[HL88] C. Huang and R. H. Litzenberger. *Foundations for Financial Economics*. Elsevier Science Publishing Co., Inc., 1988.

[HQS07] M. Hirschberger, Y. Qi, and R. E. Steuer. Randomly generating portfolio-selection covariance matrices with specified distributional characteristics. *European Journal of Operational Research*, 177(3):1610–1625, 2007.

[Inv05] Investmentgesetz (InvG). online: `http://www.bafin.de/gesetze/invg.htm`, 2005. Last accessed: 2006-10-18.

[JHLM01] N. J. Jobst, M. D. Horniman, C. A. Lucas, and G. Mitra. Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. *Quantitative Finance*, 1:489–501, 2001.

[JK80] J. D. Jobson and B. Korkie. Estimation for markowitz efficient portfolios. *Journal of the American Statistical Association*, 75(371):544–554, 1980.

[JLM05] B. I. Jacobs, K. N. Levy, and H. M. Markowitz. Portfolio optimization with factors, scenarios, and realistic short positions. *Operations Research*, 53(4):586–599, 2005.

[JM03] R. Jagannathan and T. Ma. Risk reduction in large portfolios: Why imposing the wrong constraints helps. *The Journal of Finance*, 53(4):1651–1683, 2003.

[Kal02] J. Kallrath. *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis*. Vieweg, 2002.

[KLM84] Y. Kroll, H. Levy, and H. M. Markowitz. Mean-variance versus direct utility maximization. *Journal of Finance*, 39(1):47–61, March 1984.

[KPU02] P. Krokhmal, J. Palmquist, and S. Uryasev. Portfolio optimization with Conditional Value-at-Risk objective and constraints. *The Journal of Risk*, 4(2):11–27, 2002.

[KT79] D. Kahnemann and A. Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–292, 1979.

[KT04] D. Kahnemann and A. Tversky. Prospect theory: An analysis of decision under risk. In A. Tversky and E. Shafir, editors, *Preferences, Belief, and Similarity*, pages 549–581. The MIT Press, 2004.

[KW02] H. Konno and A. Wijayanayake. Portfolio optimization under d.c. transaction consts and minimal transaction unit constraints. *Journal of Global Optimization*, 22:137–154, 2002.

[KY91] H. Konno and H. Yamazaki. Mean-absolute deviation portfolio optimization model and its application to tokyo stock market. *Management Science*, 37(5):519–531, May 1991.

[KY05] H. Konno and R. Yamamoto. Global optimization versus integer programming in portfolio optimization under nonconvex transaction costs. *Journal of Global Optimization*, 32:207–219, 2005.

[KZ83] J. G. Kallberg and W. T. Ziemba. Comparison of alternative utility functions in portfolio selection problems. *Management Science*, 29(11):1257–1276, November 1983.

[LSW06] D. Li, X. Sun, and J. Wang. Optimal lot solution to cardinality constrained mean-variance formulation for portfolio selection. *Mathematical Finance*, 16(1):83–101, 2006.

[LTZ06] M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, March 2006.

[LW01] S. F. Leroy and J. Werner. *Principles of Financial Economics*. Cambridge University Press, 2001.

[LW02] D. Lin and S. Wang. A genetic algorithm for portfolio selection problems. *Advanced Modeling and Optimization*, 4(1):13–27, 2002.

[LW04] O. Ledoit and M. Wolf. Honey, i shrunk the sample covariance matrix. *Journal of Portfolio Management*, 31(1):110–119, 2004.

[Man63] B. Mandelbrot. The variation of certain speculative prices. *The Journal of Business*, 36(4):394–419, 1963.

[Mar56] H. M. Markowitz. The optimization of a quadratic function subject to linear constraints. *Naval Research Logistics Quarterly*, 3:111–133, 1956.

[Mar59] H. M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Wiley, Yale University Press, 1959.

[Mar87] H. M. Markowitz. *Mean-Variance Analysis in Portfolio Choice and Capital Markets*. Blackwell Publishers, 1987.

[Mar02a] D. Maringer. Portfolioselektion bei Transaktionskosten und Ganzzahligkeitsbeschränkungen. *Zeitschrift für Betriebswirtschaft*, 72(11):1155–1175, 2002.

[Mar02b] D. Maringer. Werpapierselektion mittels Ant Systems. *Zeitschrift für Betriebswirtschaft*, 72(12):1221–1240, 2002.

[Mar05] D. Maringer. Distribution assumptions and risk constraints in portfolio optimization. *Computational Management Science*, 2(2):139–153, 2005.

[Mic89] R. O. Michaud. The markowitz optimization enigma: Is 'optimized' optimal? *Financial Analysts Journal*, 45(1):31–42, 1989.

[Mie98] K. Miettinen. *Nonlinear multiobjective optimization*. Kluwer, 1998.

[MK03] D. Maringer and H. Kellerer. Optimization of cardinality constrained portfolios with a hybrid local search algorithm. *OR Spectrum*, 25:481–495, 2003.

[MK06] U. Mello and I. Khabibrakhmanov. On the reusability and numeric efficiency of C++ packages in scientific computing. online: `citeseer.ist.psu.edu/634047.html`, 2006. Last accessed: 2006-10-11.

[MOS03] R. Mansini, W. Ogryczak, and M. G. Speranza. LP solvable models for portfolio optimization: A classification and computational comparison. *IMA Journal of Management Mathematics*, 14:187–220, 2003.

[MS99] R. Mansini and M. G. Speranza. Heuristic algorithms for the portfolio selection problem with minimum transaction lots. *European Journal of Operational Research*, 114:219–233, 1999.

[NEO06] Optimization software guide. online: `http://www-fp.mcs.anl.gov/otc/Guide/SoftwareGuide/`, 2006. Last accessed: 2006-10-18.

[NW99] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag New York, 1999.

[Per84] A. F. Perold. Large-scale portfolio optimization. *Management Science*, 30(10):1143–1160, October 1984.

[Rab00] M. Rabin. Risk aversion and expected-utility theory: A calibration theorem. *Econometrica*, 68(5):1281–1292, September 2000.

[Rud94] M. Rudolf. *Algorithms for Portfolio Optimization and Portfolio Insurance*. Dissertation, Hochschule Sankt Gallen, 1994.

[SBS07] M. Stein, J. Branke, and H. Schmeck. Efficient implementation of an active set algorithm for large scale portfolio selection. *Computers & Operations Research*, to appear, 2007.

[Sch02] A. Schaerf. Local search techniques for constrained portfolio selection problems. *Computational Economics*, 20(3):170–190, 2002.

[Sch06] B. Scheckenbach. Envelope-based portfolio optimization with complex constraints. Master's thesis, Institute AIFB, University of Karlsruhe, Germany, 2006.

[Sha63] W. F. Sharpe. A simplified model for portfolio analysis. *Management Science*, 9(2):277–293, 1963.

[Sim97] Y. Simaan. Estimation risk in portfolio selection: The mean variance model versus the mean absolute deviation model. *Management Science*, 43(10):1437–1446, 1997.

[SQH06] R. E. Steuer, Y. Qi, and M. Hirschberger. Portfolio optimization: New capabilities and future methods. *Zeitschrift für Betriebswirtschaft*, 76(2):199–216, 2006.

[SS04] F. Schlottmann and D. Seese. Financial applications of multi-objective evolutionary algorithms: recent development and future research directions. In C. Coello-Coello and G. Lamont, editors, *Applications of Multi-Objective Evolutionary Algorithms*, pages 627–652. World Scientific, 2004.

[SS05] F. Schlottmann and D. Seese. A hybrid heuristic approach to discrete multi-objective optimization of credit portfolios. *Computational Statistics Data Analysis*, 47(2):373–399, 2005.

[Ste98] G. W. Stewart. *Matrix Algorithms Volume 1: Basic Decompositions*. SIAM, 1998.

[Ste01] M. C. Steinbach. Markowitz revisited: Mean-variance models in financial portfolio analysis. *SIAM Review*, 43(1):31–85, 2001.

[Sun00] S. M. Sundaresan. Continuous-time methods in finance: A review and an assessment. *The Journal of Finance*, 55(4):1569–1622, August 2000.

[SUZ03] F. Streichert, H. Ulmer, and A. Zell. Evolutionary algorithms and the cardinality constrained portfolio optimization problem. In *GOR Operations Research Conference*, pages 253–260. Springer, 2003.

[SUZ04a] F. Streichert, H. Ulmer, and A. Zell. Comparing discrete and continuous genotypes on the constrained portfolio selection problem. In *Genetic and Evolutionary Computation Conference*, volume 3103 of *LNCS*, pages 1239–1250, 2004.

[SUZ04b] F. Streichert, H. Ulmer, and A. Zell. Evaluating a hybrid encoding and three crossover operators on the constrained portfolio selection problem. In *Congress on Evolutionary Computation*, volume 1, pages 932–939. IEEE Press, 2004.

[Ury00] S. Uryasev. Conditional Value-at-Risk: Optimization algorithms and applications. *Financial Engineering News*, 14:1–5, February 2000.

[vNM44] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[Wol98] L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.

[ZTL+03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.