CHALMERS | GÖTEBORG UNIVERSITY

MASTER'S THESIS

Resource allocation – a numerical study

Christoffer Strömberg

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG UNIVERSITY
Göteborg, Sweden 2011

Thesis for the Degree of Master of Science

Resource allocation – a numerical study

Christoffer Strömberg

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematical Sciences
Chalmers University of Technology and Gothenburg University
SE-412 96 Göteborg, Sweden
Göteborg, November 2011

Mathematical Sciences
Göteborg 2011

Abstract

We consider the performance of the most important algorithms for solving the continuous, separable, differentiable and convex resource allocation problem with a singly resource constraint and lower and upper bounds on the variables. We take advantage of earlier studies to modify the relaxation algorithm such that it theoretical performs better than the earlier relaxation algorithms. In the numerical study we consider large scale problems and we compare four relaxation algorithms whereof our modified algorithms perform best, and four breakpoint algorithms whereof a generalization of an algorithm applied on the quadratic knapsack problem performs best. We finish up with comparing the best performing relaxation algorithm, breakpoint algorithm and a quasi-Newton method. We conclude that our modification of the relaxation algorithm performs best in most cases. Moreover, the result implies that the practical time complexity for the breakpoint algorithm and the relaxation algorithm is linear.

Acknowledgments

Thanks to my tutor Michael Patriksson for advices, guidance and music. Thanks to my dearest Sanna Larsson for endurance, care and love . Thanks to Daniel Johansson for motivation and puns. Thanks to Svante for fireworks and guitar solos. Thanks to friends, family, Pink Floyd, Lou Reed, Steinbeck, Söderberg, Chalmers and animals.

Contents

1 Introduction

In this thesis the most successful algorithms for solving a separable, convex, continuous and differentiable resource allocation problem with a single inequality or equality constraint are evaluated. A set of common objective functions are set up and an exhaustive evaluation is made of the algorithms.

The problem formulated as in the survey [Pat08], that is the origin of this approach: Let $\phi_j : \mathbb{R} \to \mathbb{R}$ and $g_j : \mathbb{R} \to \mathbb{R}$ be convex and differentiable. Moreover, let $b \in \mathbb{R}$ and $-\infty \leq l_j < u_j \leq \infty$. Consider the problem:

$$\underset{x}{\text{minimize}} \quad \phi(\mathbf{x}) := \sum_{j=1}^{n} \phi_j(x_j), \tag{1a}$$

$$\text{subject to} \quad g(\mathbf{x}) := \sum_{j=1}^{n} g_j(x_j) \leq b, \tag{1b}$$

$$l_j \leq x_j \leq u_j, \quad j = 1, ..., n. \tag{1c}$$

We also consider the similar problem, where the inequality of constraint (1b) is replaced by equality, i.e.

$$\underset{x}{\text{minimize}} \quad \phi(\mathbf{x}) := \sum_{j=1}^{n} \phi_j(x_j), \tag{2a}$$

$$\text{subject to} \quad g(\mathbf{x}) := \sum_{j=1}^{n} a_j x_j = b, \tag{2b}$$

$$l_j \leq x_j \leq u_j, \quad j = 1, ..., n, \tag{2c}$$

where $a \neq 0$ and of the same sign. We assume that there exist a solution and that it is bounded such that $-\infty < \phi^* < \infty$. We define $\mathbf{J} := \{1, 2, ..., n\}$.

Section 1 considers the background of the thesis and the importance of having access to fast algorithms solving the resource allocation problems (1) and (2). The most important earlier works which motivate this thesis are outlined.

Section 2 described the necessary theory behind the algorithms. First, the essential theory of Lagrange relaxation and the Karush–Kuhn–Tucker (KKT) conditions are briefly treated. This follows by the concepts of relaxation algorithms, four relaxation algorithms are posted. Further, the concepts of breakpoint algorithms are described and primary four versions of breakpoint algorithms are posted. Finally, we post a quasi-Newton method for solving problem (2).

Section 3 describes the procedure for the numerical experiments; the test problem set is specified. Methods for evaluate performance of similar algorithms in earlier studies are reviewed and a performance profile is defined. We also specify the computer language and the computer used in the numerical experiments.

In Section 4, we show the results from the numerical experiments. The structure is such that, first we compare the relaxation algorithms. Second we compare the breakpoint algorithms and third we compare the best performing relaxation algorithm, the best performing breakpoint algorithm and the quasi-Newton method. Finally we give some hints of future work and make an overall conclusion.

In Appendix A, we show the cpu-times from the numerical experiments of the best performing algorithms.

1.1 Objective

This thesis have its origin in Patriksson's survey on the continuous nonlinear resource allocation problem [Pat08]. Since the first algorithm reported in 1957 ([CAA57]), many algorithms for solving the singly constrained resource allocation problem have been proposed, referring to the significant bibliography in [Pat08]. Many of these might be out of date in the sense of computational time and there are algorithms that have been ignored or forgotten. Hence, the objective is to evaluate and develop the most important algorithms in the bibliography of [Pat08], of course we also consider articles published after the publication of [Pat08]. From the evaluation we establish the best performing algorithm solving problems (1) and (2).

1.2 Motivation

The separable, convex, continuous and differentiable resource allocation problem with a singly resource constraint is very common and occurs in many different areas, e.g. search theory [Koo99], economics [Mar52], stratified sampling [BRS99], inventory systems [MaK93] and queuing manufacturing networks[BiT89]. Further, the continuous resource allocation problem occurs as subproblem in algorithms that effectively solve the integer resource allocation problem [BrS02], [IbK88, p.p. 72-75]. Moreover, problem (1) and (2) can be used as a subproblem when solving resource allocation problems with more than one resource constraint [FeZ83]. Also, problem (1) and (2) are used as a subproblem to solve (1) and (2) with a nonseparable objective function ϕ [DSV07].

Efficient algorithms are required, since the problems (1) and (2) often are very large and in the case where the problems are used as subproblems, it has to be resolved several times. Next, a naive example clarifies the complications that occur when a problem grows large.

Example 1. Suppose we want to search for a treasure. We know that the treasure is buried on one of n islands. Additionally, we know that the treasure is on island j with a specific probability m_j and our only constraint is that we have an excavator with petrol for b days. If we search x_j days on island j we will find the treasure with the probability $1 - e^{-b_j x_j}$ where b_j describes the terrain on island j (the difficulty of finding the treasure on island j). With this information the problem may be stated as follows:

$$\underset{x}{\text{maximize}} \sum_{j=1}^{n} m_j(1 - e^{-b_j x_j}), \tag{3a}$$

$$\text{subject to} \sum_{j=1}^{n} x_j = b, \tag{3b}$$

$$0 \leq x_j \leq u_j,\ j = 1, 2, ..., n. \tag{3c}$$

where u_j is the time for exploring the entire island j. This problem is easily solved with Matlab by the instrinct function *fmincon*. But when n grows large the CPU-time tends to be large. Table 1 shows CPU-time for *fmincon* (Setup to use interior-point) and the CPU-time for a simple implementation of the classical Bitran–Hax–algorithm, [BiH81].

n	CPU-time Bitran Hax [s]	CPU-time fmincon [s]
100	0.00042	0.47
200	0.00062	1.62
300	0.00092	2.96
400	0.00066	5.84
500	0.00084	9.55
1000	0.0012	50.74

Table 1: CPU-time for *fmincon* and Bitran–Hax–algorithm [BiH81] implemented in Matlab. Each CPU-time reported is the mean value of 5 computations of problem (3). The parameters a_j and b_j were chosen randomly with Matlabs function *rand()* and $b = 1$.

From Table 1 it is realized that Matlab's predefined function *fmincon* solves problem (3) relatively slow. Moreover, while Bitran's and Hax's algorithm gives the optimal solution, *fmincon* only gives an approximal solution.

(*fmincon* uses an interior point method that standardize the inequality constraints by adding a slack variable and add a penalty function to the objective. Then *fmincon* tries to fulfill the optimality conditions (KKT) by linearization of the KKT–conditions (Newton step). If this do not succeeds *fmincon* uses a conjugate gradient method, i.e. the approximated function is approximated by a quadratic function and the constraint are linearized in a trust region.)

Many numerical studies in the field of resource allocation problem has been done, for example see [KoL98], [NiZ92], [Kiw07], [BiH81], [Kiw08], [Kiw08b] and [RJL92]. Beside [Kiw08], [Kiw08b] and [Kiw07] where the quadratic knapsack problem is studied, none of the earlier approaches studies the large scaled version of problem (1) and (2). And there are, especially two algorithms [Ste01], [NiZ92, Section 1.4] that according to the authers are very promising that have not been evaluated properly for problems (1) and (2). Hence, a review of these two algorithms is needed. Moreover, only one earlier study [KoL98] evaluate different kind of percentage of the solution that equals the lower or upper bounds (see Section 3.2), and this is only done for $n = 10,000$. From this we conclude that more numerical experiments is needed to explore the algorithms for solving problems (1) and (2).

1.3 Limitations

For the algorithms, we assume that the dual variable can be found in a closed form. If that is not the case we need to implement a numerical method in some of the steps e.g. a Newton method. We only consider customized algorithms for the problem in hand since we assume that they performs better, in the sense of CPU-time, than more general algorithms.

Due to time, we consider five different objective functions ϕ in the numerical study. For all problems in the problem set we consider linear resource constraints since a non-linear resource constraint is not valid for problem (2) if we want it to be convex.

2 Algorithms

The first article which considering customized algorithms for the continuous resource allocation problem was published 1957, [Pat08]. Since then, several algorithm have been developed for solving special cases of problem (1) and (2) especially with the linear resource constraint $g_j(x_j) = x_j$. The publication of such a huge amount of articles concerning the resource allocation problem makes perfect sense since a small modification might imply a great improvement, concerning to CPU-time etc. Many of the algorithms are very similar, but there are mainly three fundamental algorithms, namely;

- Breakpoint methods; from the lower and upper bounds (1c) or (2c), a set of breakpoints for the dual variable can be determined. From the optimality conditions and the breakpoints, it is possible to determine if a variable equals the lower or upper bound or not. So by a search among the breakpoints it is possible to find an optimal solution.

- Relaxation methods; for problem (1) or (2), the lower and upper bounds (1c) or (2c) are relaxed. The relaxed problem is solved and the lower and upper bounds are considered again. If the optimality conditions are satisfied, the solution is optimal; otherwise we fix either the violated lower or upper bounds and resolve the relaxed problem.

- Numerical methods; in this approach a Newton-like linesearch method is applied; The resource constraint (1b) or (2b) is evaluated as a function of the dual variable. And since the dual variable belongs to $\mathbb{R}$, it is possible to apply a linesearch to find an approximal optimal solution.

In Section 2.1, we introduce some essential properties for the problems (1) and (2), the KKT conditions are posted and from these an optimal solution is proposed.

Section 2.2, features the relaxation algorithm. Two algorithms similar to earlier published algorithms and two modifications of these are post.

Section 2.3, contains mainly four breakpoint algorithms.

In Section 2.4, we present a quasi-Newton method for solving problem (2). This method have been successful for the quadratic case in parallel computations [NiZ92].

2.1 Basic properties

The analysis of problems (1) and (2) is quite similar, but there are some fundamental difference that will affect the algorithms for each problem. In

Section 2.1.1 we treat the inequality problem i.e. problem (1) and in Section 2.1.2, we briefly point out what differs for the equality problem i.e. problem (2).

2.1.1 Inequality constraint

To begin with consider the Lagrange function of problem (1):

$$L(\mathbf{x}, \mu, \rho, \lambda) = \sum_{j=1} \phi_j(x) + \mu \left(\sum_{j=1} g_j(x_j) - b \right) + \sum_{j=1} \rho_j(l_j - x_j) + \sum_{j=1} \lambda_j(x_j - u_j),$$

where $\mu \in \mathbb{R}_+$ and $\rho, \lambda \in \mathbb{R}^n_+$ are the dual variables associated with constraint (1b) and (1c), respectively. The Lagrange multiplier refers μ^*, ρ^* and λ^* are defined for the optimal solution ϕ^* such that $\phi^* \geq \inf_{x \in \mathbf{X}} L(x, \mu^*, \rho^*, \lambda^*)$. Hence, instead of finding a solution to problem (1) we may find a lower bound to ϕ^* due to the Lagrangian relaxation of the problem, that is to

$$\underset{x}{\text{minimize}} \;\; L(x, \mu^*, \rho^*, \lambda^*). \tag{4}$$

We need to find the optimal dual variables μ^*, ρ^* and λ^*. We introduce the Lagrangian dual function $q(\mu, \rho, \lambda) := \min_{x \in \mathbb{R}^n} \;\; L(x, \mu, \rho, \lambda)$ and the dual problem takes the form:

$$\underset{\mu \geq 0;\, \rho, \lambda \geq 0^n}{\text{maximize}} \quad q(\mu, \rho, \lambda). \tag{5}$$

Further, if we assume that there exists a point $\mathbf{x}_0$ to problem (1) such that

$$g(\mathbf{x}_0) < 0 \quad (\text{Slater's CQ})$$

holds then there exists an optimal solution q^* to the dual problem (5) such that $\phi^* = q^*$. This follows from the convexity of the problem and the assumption that the optimal solution of problem (1) is bounded [AEP05, Theorem 5.45].

However, we need to find the solution of problem (4) and we know from fundamental calculus that if the derivative of a convex function equals zero we have found an optimum. In other words we want the gradient for the primal and dual variables in $L(x, \mu, \rho, \lambda)$ to equal zero. The optimality conditions for $L(x, \mu, \rho, \lambda)$ can be stated as

$$\phi_j'(x_j) + \mu g_j'(x_j) - \rho_j + \lambda_j = 0, \quad \text{for } j = 1, 2, ..., n, \tag{6a}$$

$$\mu \left(\sum_{j=1} g_j(x_j) - b \right) = 0, \tag{6b}$$

$$\lambda_j(x_j - u_j) = 0, \quad \text{for } j = 1, 2, ..., n, \tag{6c}$$

$$\rho_j(l_j - x_j) = 0, \quad \text{for } j = 1, 2, ..., n. \tag{6d}$$

Additional, we know from the dual problem (5) and primal problem (1) that

$$\mu \in \mathbb{R}_+, \tag{7a}$$

$$\rho, \lambda \in \mathbb{R}^n_+, \tag{7b}$$

$$\sum_{j=1} g_j(x_j) \leq b, \tag{7c}$$

$$l_j \leq x_j \leq u_j, \quad \text{for } j = 1, 2, ..., n. \tag{7d}$$

The optimality conditions (6a)–(6d) and the feasibility conditions (7a)–(7d) are in fact the KKT conditions for problem (1). If a convex problem is considered, the KKT conditions are sufficient for global optimality [AEP05, Theorem 5.45].

Assume first that μ>0, then constraint (7c) must hold with equality according to (6b). Without loss of generality we assume that $l_j < u_j$ since if $l_j = u_j$ we can set $x^*_j = l_j = u_j$ and reduce the problem. The optimality conditions (6a)–(6d) and (7a)–(7d) left us with three very interesting cases:

First, suppose that $l_j < x_j < u_j$; then, for conditions (6c) and (6d) to hold we have that $\rho_j = \lambda_j = 0$. From condition (6a), we determine μ such that

$$\mu g'_j(x_j) = -\phi'_j(x_j) \tag{8}$$

holds. Notice that if we can fulfill condition (7c) with equality, all the KKT conditions (6a)–(6d) and (7a)–(7d) are fulfilled.

Second, suppose now that $x_j = l_j$, then (6c) implies that $\lambda_j = 0$ must hold. Assume that $\rho_j \in \mathbb{R}_+$; From (6a), we have that

$$0 \leq \rho_j = \phi'_j(l_j) + \mu g'_j(l_j). \tag{9}$$

Hence,

$$\mu^* g_j(l_j) \geq -\phi'_j(l_j). \tag{10}$$

Similar to above, if we can fulfill condition (7c) with equality, all the KKT conditions (6a)–(6d) and (7a)–(7d) are fulfilled.

Finally, assume that $x_j = u_j$, then (6d) implies that $\rho_j = 0$ must hold. Assume that $\lambda_j \in \mathbb{R}_+$; From (6a), we then have that

$$0 \geq -\lambda_j = \phi'_j(u_j) + \mu g'_j(u_j). \tag{11}$$

Hence,

$$\mu^* g_j(u_j) \geq -\phi'_j(u_j). \tag{12}$$

Again, if we can fulfill condition (7c) with equality, all the KKT conditions (6a)–(6d) and (7a)–(7d) are fulfilled.

To summarize, let $\bar{x}_j(\mu)$ be the solution of (6a) when ρ_j and $\lambda_j = 0$. Further, if we can find μ^* such that $\sum_{j=1} g_j(x_j(\mu^*)) = b$ where $x_j(\mu)$ is defined as the solution of (4) for a given μ, then we can find the optimal solution from

$$x_j(\mu) = \begin{cases} \bar{x}_j(\mu), & \text{if } \mu g_j'(\bar{x}_j) = -\phi_j'(\bar{x}_j), \\ l_j, & \text{if } \mu g_j'(l_j) \geq -\phi_j'(l_j), \\ u_j, & \text{if } \mu g_j'(u_j) \leq -\phi_j'(u_j), \end{cases} \tag{13}$$

$$\rho_j(\mu) = \begin{cases} \phi_j'(l_j) +_j (l_j), & \text{if } \mu g_j'(l_j) \geq -\phi_j'(l_j), \\ 0, & \text{if } \mu g_j'(l_j) < -\phi_j'(l_j), \end{cases} \tag{14}$$

$$\lambda_j(\mu) = \begin{cases} -\phi_j'(u_j) - \mu g_j(u_j), & \text{if } \mu g_j'(u_j) \leq -\phi_j'(u_j), \\ 0, & \text{if } \mu g_j'(u_j) > -\phi_j'(u_j), \end{cases} \tag{15}$$

or

$$x_j(\mu) = \begin{cases} \bar{x}_j(\mu), & \text{if } \mu = -\phi_j'(\bar{x}_j)/g_j'(\bar{x}_j), \\ l_j, & \text{if } \mu \geq -\phi_j'(l_j)/g_j'(l_j), \\ u_j, & \text{if } \mu \leq -\phi_j'(u_j)/g_j'(u_j), \end{cases} \tag{16}$$

$$\rho_j(\mu) = \begin{cases} \phi_j'(l_j) + \mu g_j(l_j), & \text{if } \mu \geq -\phi_j'(l_j)/g_j'(l_j), \\ 0, & \text{if } \mu < -\phi_j'(l_j)/g_j'(l_j), \end{cases} \tag{17}$$

$$\lambda_j(\mu) = \begin{cases} -\phi_j'(u_j) - \mu g_j(u_j), & \text{if } \mu \leq -\phi_j'(u_j)/g_j'(u_j), \\ 0, & \text{if } \mu > -\phi_j'(u_j)/g_j'(u_j). \end{cases} \tag{18}$$

Theorem 1. If $\mu^* > 0$ for problem (1) then $x_j(\mu)$, $\rho_j(\mu)$ and $\lambda(\mu)$ determined from (16)–(18) satisfy all KKT conditions besides (6b) and (7c).

(Equivalently: For problem (2), $x_j(\mu)$, $\rho_j(\mu)$ and $\lambda(\mu)$ determined from (16)–(18) satisfy all KKT conditions besides (6b) and (7c).)

Proof. Conditions (6a), (6c), (6d), (7a), (7d) is trivially satisfied from the discussion above. Consider condition (7b), we start with showing that $\rho_j \in \mathbb{R}_+$ for all $j = 1, 2, ..., n$. If $\mu < -\phi_j'(l_j)/g_j'(l_j)$ then $\rho_j = 0$ from (17). If $\mu \geq -\phi_j'(l_j)/g_j'(l_j)$ then from (17) we have $\rho_j = \phi_j'(l_j) + \mu g_j(l_j) \geq \phi_j'(l_j) - \phi_j'(l_j) = 0$ from (14). The proof for $\lambda_j \in \mathbb{J}$ for all $j = 1, 2, ..., n$ is similar. □

So the problem is reduced to finding μ such that $\sum_{j=1}^{n} g_j(x_j(\mu)) = b$, assumed that $\mu^* > 0$. This is in fact nothing else than to solve the dual problem (5). To see this consider the derivative of q

$$q'(\mu) = \sum_{j=1} g_j(x_j(\mu)) - b. \tag{19}$$

If we like to determine $x_j(\mu)$ from (16) several times for different μ. It would be a waste of time to determine $\phi_j'(l_j)/g_j'(l_j)$ and $\phi_j'(u_j)/g_j'(u_j)$ every time. So it make sense to define a set of lower and upper breakpoints such that:

Definition 2 (Lower breakpoints). $v^l = \left\{ \frac{-\phi_j'(l_j)}{g_j'(l_j)} \right\}_{j=1,2,\ldots,n}$

Definition 3 (Upper breakpoints). $v^u = \left\{ \frac{-\phi_j'(u_j)}{g_j'(u_j)} \right\}_{j=1,2,\ldots,n}$

Assume now that μ = 0 is feasible, than we note that condition (7c) do not have to be fulfilled with equality.

Theorem 4. If $\mu = 0$ is feasible for problem (1), then $\mu^* = 0$ is optimal.

Proof. Assume $\mu = 0$ is feasible i.e. we have $\mathbf{x}(\mu)$ such that $\sum_{j=1} g_j(x_j(\mu)) \leq b$. Hence, (6b), (7a) and (7c) are satisfied trivially. Conditions (6a), (6c), (6d) and (7d) are satisfied from (13), (14) and (15).

Hence, we show that $\rho_j,\ \lambda_j \in \mathbb{R}_+$ for all j: For $x_j > l_j$ we have $\rho_j = 0 \in \mathbb{R}_+$. For $x_j = l_j$ we have from (14) that $\mu^* g_j'(l_j) = 0 \geq -\phi_j'(l_j)$ implies $0 \leq \phi_j'(l_j) = \rho_j \in \mathbb{R}_+$. Similar we can show that $\lambda_j \in \mathbb{R}_+$ for all j.

□

In Section 2.2.2, we will see that it is possible to determine the optimal solution x^* in a closed form for some special cases of problem (1).

2.1.2 Equality constraints

Let us now consider problem (2) where the inequality of the primal constraint is exchanged by an equality. For the problem to be convex, the resource constraint (2b) have to be affine such that constraint (7c) becomes:

$$\sum_{j \in \mathbf{J}} g_j(x_j) := \sum_{j \in \mathbf{J}} a_j x_j.$$

Additional, we will assume that ther exists a point $\mathbf{x}_0$ such that $\sum_{j \in \mathbf{J}} a_j x_j = b$ (Slater's CQ) holds. Beside the resource constraint, the Lagrangian and the KKT conditions will take the same form as in section 2.1.1 but with one important difference, we will allow $\mu < 0$ i.e.

$$\mu \in \mathbb{R}.$$

2.1.3 The pegging process

Assume that we have a lower μ_l and an upper limit μ_u for the optimal dual variable μ such that $\mu_l \leq \mu^* \leq \mu_u$. Define $\mathbf{J}_- := \{j : \mu_l \geq -\phi_j'(l_j)/g_j'(l_j)$ for all $j \in \mathbf{J}\}$ and $\mathbf{J}_+ := \{j : \mu_u \leq -\phi_j'(u_j)/g_j'(u_j)$ for all $j \in \mathbf{J}\}$. Let

$\mathbf{J}^k := \mathbf{J} \setminus \{\mathbf{J}_- \cup \mathbf{J}_+\}$ and let $b^k = b - \sum_{j \in \mathbf{J}_-} g_j(l_j) - \sum_{j \in \mathbf{J}_+} g_j(u_j)$. Now we can define a subproblem of problem (1) such that

$$\underset{x}{\text{minimize}} \quad \phi(\mathbf{x}) := \sum_{j \in \mathbf{J}^k} \phi_j(x_j), \tag{20a}$$

$$\text{subject to} \quad g(\mathbf{x}) := \sum_{j \in \mathbf{J}^k} g_j(x_j) \leq b^k, \tag{20b}$$

$$l_j \leq x_j \leq u_j, \quad j \in \mathbf{J}^k. \tag{20c}$$

Similarly we can define a subproblem of problem (2)

$$\underset{x}{\text{minimize}} \quad \phi(\mathbf{x}) := \sum_{j \in \mathbf{J}^k} \phi_j(x_j), \tag{21a}$$

$$\text{subject to} \quad g(\mathbf{x}) := \sum_{j \in \mathbf{J}^k} a_j x_j = b^k, \tag{21b}$$

$$l_j \leq x_j \leq u_j, \quad j \in \mathbf{J}^k. \tag{21c}$$

Consider problem (20) and assume that $\mu^* > 0$ which, from Section 2.1.1, implies that constraint (1b) has to be fulfilled with equality. Hence, the reasoning for problem (20) will be analogue to problem (21) besides we allow $\mu < 0$ for problem (21).

For any given dual variable μ^k we can determine the primal solution $\mathbf{x}^k$ of problem (20) from (16). We know from Section 2.1.1 that all KKT conditions beside the resource constraint (7c) is satisfied. Substituting $\mathbf{x}^k$ into the resource constraint leaves us with 3 cases, namely

$$\sum_{j \in \mathbf{J}^k} g_j(x_j^k) = b^k, \tag{22a}$$

$$\sum_{j \in \mathbf{J}^k} g_j(x_j^k) < b^k, \tag{22b}$$

$$\sum_{j \in \mathbf{J}^k} g_j(x_j^k) > b^k. \tag{22c}$$

Evaluate the optimality from (22a)–(22c) will be referred to as explicit evaluation.

If (22a) is fulfilled for $\mathbf{x}^k$ than all KKT conditions are met and the optimal solution $\mathbf{x}^* = \mathbf{x}^k$. On the other hand, what if (22b) is fulfilled? Clearly $\mathbf{x}^k$ is not feasible but we know that $\mathbf{x}^*$ is such that $\sum_{j \in \mathbf{J}^k} g_j(x_j^*) > \sum_{j \in \mathbf{J}^k} g_j(x_j^k)$.

But the only thing we know about g_j is that it is convex and differentiable. That is not enough since g_j can be increasing in one interval and decreasing in another, e.g. $g_j(x_j) = x_j^2$. Hence, we need g_j to be either monotonically increasing or monotonically decreasing. For problem (1), Bretthauer and Shetty [BrS95] considered four cases equivalent to:

Case 1: For all $j \in \mathbf{J}$; g_j is decreasing and $-\phi_j(x_j)/g_j(x_j)$ is increasing in x_j.

Case 2: For all $j \in \mathbf{J}$; g_j is increasing and $-\phi_j(x_j)/g_j(x_j)$ is decreasing in x_j.

Case 3: For all $j \in \mathbf{J}$; g_j is decreasing and $-\phi_j(x_j)/g_j(x_j)$ is decreasing in x_j.

Case 4: For all $j \in \mathbf{J}$; g_j is increasing and $-\phi_j(x_j)/g_j(x_j)$ is increasing in x_j.

The property that $-\phi_j(x_j)/g_j(x_j)$ is increasing in x_j implies that $x(\mu)$ determined from (16) is nondecreasing and vice versa.

Remark 1. For problem (2), only Cases 1 and 2 are feasible since $g_j(x_j) = a_j x_j$ where $a_j < 0$ for all j respectively $a_j > 0$ for all j. Cases 3 and 4 are not feasible for problem (2) since if $g_j(x_j) = a_j x_j$ where $a_j < 0$ for all j and $x_j(\mu)$ is decreasing then ϕ_j must be concave since $\mu(x_j) = -\phi_j'(x_j)/a_j$, but ϕ' is convex.

If we assume that one of the Cases 1, 2, 3 or 4 holds, we can state the following theorems about pegging. First, define $\mathbf{J}_+^k := \{j : \mu^k \leq -\phi_j'(u_j)/g_j'(u_j)$ for $j \in \mathbf{J}^k\}$.

Theorem 5. If Case 2 holds for problem (1), if $\mu^* > 0$ and if (22b) holds, then we can peg all $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$.

(Equivalently; If Case 2 holds for problem (2) and if (22b) holds, then we can peg all $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$.)

Proof. From (22b) we have that $\sum_{j \in \mathbf{J}^k} g_j(x_j(\mu^k)) < b^k$. From Case 2, g_j is increasing and $x_j(\mu^k)$ is nonincreaing for all j which implies that $g_j(x_j(\mu))$ is nonincreasing in μ for all j. Hence, we have that $\mu^k \geq \mu^*$ which implies that $x_j^k \leq x_j^*$ for all j since $x_j(\mu)$ is nonincreasing in μ for all j. Hence, for $j \in \mathbf{J}_+^k$, we can peg $x_j^{k+1} = u_j = x_j^*$. □

Since we know that the optimal solution $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$, we can reduce problem (20) such that $\mathbf{J}^{k+1} = \mathbf{J}^k \setminus \mathbf{J}_+^k$. Note that we have used a part of the resource b^k, hence $b^{k+1} = b^k - \sum_{j \in \mathbf{J}_+^k} g_j(u_j)$.

Further, define $\mathbf{J}_-^k := \{j : \mu^* \geq -\phi_j'(u_j)/g_j'(u_j)$ for $j \in \mathbf{J}^k\}$. We formulate the following theorem similar to Theorem 5:

Theorem 6. If Case 2 holds for problem (1), if $\mu^* > 0$ and if (22c) holds, then we can peg all $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$.

(Equivalent; if Case 2 holds for problem (2) and if (22c) holds, then we can peg all $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$.)

Proof. The proof is similar to the proof of Theorem 5. □

Similarly to above we know that the optimal solution $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$. Hence, we can reduce problem (20) such that $\mathbf{J}^{k+1} = \mathbf{J}^k \setminus \mathbf{J}_-^k$ and $b^{k+1} = b^k - \sum_{j \in \mathbf{J}_-^k} g_j(l_j)$.

An analogue reasoning will lead to the following theorems which can be proved similar to Theorem 5.

Theorem 7. If Case 1 holds for problem (1), $\mu^* > 0$ and if (22b) holds, then we can peg all $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$.

Theorem 8. If Case 1 holds for problem (1), $\mu^* > 0$ and if (22c) holds, then we can peg all $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$.

If Case 3 or 4 holds, Bretthauer and Shetty establish that it is possible to find a closed form of the optimal solution of problem (1). They states the following theorem, (see [BrS02, Proposition 10]):

Theorem 9. (*i*) If Case 3 holds, then the optimal solution to problem (1) is $x_j^* = u_j$ for all $j \in \mathbf{J}$.
(*ii*) If Case 4 holds, then the optimal solution to problem (1) is $x_j^* = l_j$ for all $j \in \mathbf{J}$.

2.2 Relaxation algorithms

The first recursive relaxation algorithm [San71] was published 1971 [Pat08] by Sanathan. The algorithm consider problem (2) with a strictly convex and differentiable objective function and linear resource constraint $g(\mathbf{x}) := \sum_{j=1}^n a_i x_i = b$. However, Sanathan's algorithm includes both sorting of $2n$ breakpoints and an iterative solving of problem (2). Bitran and Hax [BiH81] presented a relaxation algorithm that had no need for sorting. They considered problem (2) with ϕ_j convex and differentiable and $g_j(x_j) = x_j$ for $j = 1, 2, ..., n$.

In Section 2.1.3, we post the fundamental concept of relaxation alorithms. In Section 2.2.2, we post an algorithm similar to Bretthauer's and Shetty's algorithm in [BrS02, Section 2]. In Section 2.2.3 we post an algorithm similar to the algorithm in [Ste01]. In Sections 2.2.4 and 2.2.5, we present two new modifications of the relaxation algorithm.

2.2.1 The concept of relaxation algorithms

For problem (20), we assume that $\mu^* > 0$ which, from Section 2.1.1, implies that constraint (20b) has to be fulfilled with equality. The reasoning for problem (20) will be analogue to problem (21) besides we allow $\mu < 0$ for problem (21).

The idea behind a relaxation algorithm is to solve problem (20) with relaxed lower and upper bounds. If the solution of the relaxed problem is a feasible solution for (20) then it is optimal for problem (20). Otherwise we peg variables that violate the lower or upper bounds and resolves the relaxed problem. The relaxed problem is described by:

$$\underset{x}{\text{minimize}} \quad \phi(\mathbf{x}^k) := \sum_{j \in \mathbf{J}^k} \phi_j(x_j^k), \tag{23a}$$

$$\text{subject to} \quad g(\mathbf{x}^k) := \sum_{j \in \mathbf{J}^k} g_j(x_j^k) \leq b^k, \tag{23b}$$

Since we assume that $\mu^* > 0$, the solution of problem (23) is found from the KKT conditions:

$$\phi_j'(x_j) + \mu g_j'(x_j) = 0 \quad j \in \mathbf{J}^k, \tag{24a}$$

$$\mu \left(\sum_{j \in \mathbf{J}^k} g_j(x_j^k) - b \right) = 0, \tag{24b}$$

$$\mu \in \mathbb{R}_+, \tag{24c}$$

$$\sum_{j \in \mathbf{J}^k} g_j(x_j^k) \leq b. \tag{24d}$$

Let $\hat{x}_j^k(\mu^k)$ denote the optimal solution of the relaxed problem (23). Next we consider the lower and upper bounds again. So for a moment we ignore the resource constraint and evaluate $\mathbf{x}^k$ for the following expressions similar to (16)–(18);

$$x_j^*(\mu^*) = \begin{cases} \hat{x}_j(\mu^*), & \text{if } l_j < \hat{x}_j(\mu^*) < u_j, \\ l_j, & \text{if } \hat{x}_j \leq l_j, \\ u_j, & \text{if } \hat{x}_j \geq u_j, \end{cases} \tag{25}$$

$$\rho_j^*(\mu^*) = \begin{cases} \phi_j'(l_j) + \mu^* g_j(l_j), & \text{if } \hat{x}_j \leq l_j, \\ 0, & \text{if } \hat{x}_j > l_j, \end{cases} \tag{26}$$

$$\lambda_j^*(\mu^*) = \begin{cases} -\phi_j'(u_j) - \mu^* g_j(u_j), & \text{if } \hat{x}_j \geq u_j, \\ 0, & \text{if } \hat{x}_j < u_j. \end{cases} \tag{27}$$

One can easily show that x_j^*, ρ_j^* and λ_j^* defined by (25)–(27) satisfies all the KKT condition for problem (20) beside condition (7c) and (6b), for proof see [BrS95, proposition 1-5]. It is also possible to see that relation (16) is equivalent to (25) by letting $\Phi_j(x_j) := \phi_j'(x_j)/g_j'(x_j)$ operate on the conditions in (25). It follows from $\Phi_j(\hat{x}_j(\mu)) = -\mu$, $\Phi_j(l_j) = \phi_j'(l_j)/g_j'(l_j)$ and $\Phi_j(u_j) = \phi_j'(u_j)/g_j'(u_j)$.

To check if the solution generated by (25) is feasible for problem (20), we have to evaluate the resource constraint. This can be done from the explicit evaluation (22a)–(22c) but for relaxation algorithms a similar method to evaluate $\mathbf{x}^k$ is used frequently. For problem (23), we note that $\hat{\mathbf{x}}^k$ will satisfy the resource constraint with equality. When we determine $\mathbf{x}^k$ from $\hat{\mathbf{x}}^k$ we set x_j to either l_j or u_j if $\hat{x}_j$ violate the lower or upper bound and if not $x_j = \hat{x}_j$. Hence, if we assume that g_j is either decresing for all $j \in \mathbf{J}^k$ or increasingfor all $j \in \mathbf{J}^k$, we can evaluate the resource constraint by consider the summation of the violated lower and upper bounds. Define Δ^k and ∇^k such that

$$\Delta^k = \sum_{j \in \mathbf{J}_+^k} \left(g_j(\hat{x}_j^k) - g(u_j) \right), \tag{28a}$$

$$\nabla^k = \sum_{j \in \mathbf{J}_-^k} \left(g(l_j) - g(\hat{x}_j^k) \right), \tag{28b}$$

where $\mathbf{J}_+^k = \{j : \hat{x}_j^k > u_j \text{ for } j \in \mathbf{J}^k\}$ and $\mathbf{J}_-^k = \{j : \hat{x}_j^k < l_j \text{ for } j \in \mathbf{J}^k\}$. We can then evaluate $\mathbf{x}^k$ from

$$\Delta^k = \nabla^k, \tag{29a}$$

$$\Delta^k > \nabla^k, \tag{29b}$$

$$\Delta^k < \nabla^k. \tag{29c}$$

Evaluate the optimality from (29a)–(29c) will be referred to as implicit evaluation.

Theorem 10. If g_j is monotonically increasing for all $j \in \mathbf{J^k}$ or if g_j monotonically decreasing for all $j \in \mathbf{J^k}$, then the explicit evaluation (22a)–(22c), is equivalent to the implicit evaluation (29a)–(29c) i.e. (22a) $\Leftrightarrow$ (22a), (29b) $\Leftrightarrow$ (22b) and (29c) $\Leftrightarrow$ (22c).

Proof. First assume that for all $j \in \mathbf{J}^k$, g_j is increasing. Then we have:

$$\begin{aligned}
\sum_{j\in\mathbf{J}^k} g_j(x_j^k) = & \sum_{j\in\mathbf{J}^k\setminus\{\mathbf{J}_+^k\cup\mathbf{J}_-^k\}} g_j(\hat{x}_j^k) + \sum_{j\in\mathbf{J}_-^k} \left\{g_j(l_j) - g_j(\hat{x}_j^k) + g_j(\hat{x}_j^k)\right\} \\
& + \sum_{j\in\mathbf{J}_+^k} \left\{g_j(u_j) - g_j(\hat{x}_j^k) + g_j(\hat{x}_j^k)\right\} \\
= & \sum_{j\in\mathbf{J}^k\setminus\{\mathbf{J}_+^k\cup\mathbf{J}_-^k\}} g_j(\hat{x}_j^k) + \sum_{j\in\mathbf{J}_-^k} \left\{g_j(l_j) - g_j(\hat{x}_j^k)\right\} + \sum_{j\in\mathbf{J}_-^k} g_j(\hat{x}_j^k) \\
& - \sum_{j\in\mathbf{J}_+^k} \left\{g_j(u_j) - g_j(\hat{x}_j^k)\right\} + \sum_{j\in\mathbf{J}_+^k} g_j(\hat{x}_j^k) \\
= & \sum_{j\in\mathbf{J}^k} g_j(\hat{x}_j^k) + \nabla^k - \Delta^k \\
= & b^k + \nabla^k - \Delta^k
\end{aligned}$$

where the last equality follows from the fact that $\hat{\mathbf{x}}^k$ is the solution of problem (23). Hence, $\hat{\mathbf{x}}^k$ must satisfy the resource constraint (23b). From (28a) and (28b) we know that $\Delta^k, \nabla^k \geq 0$ since g_j is increasing. Hence, if $\Delta^k = \nabla^k$ then (22a) holds, if $\Delta^k > \nabla^k$ then (22b) holds, and if $\Delta^k < \nabla^k$ then (22c) holds. The proof for g_j decreasing for all $j \in \mathbf{J}$ is similar. □

Example 2. Consider the quadratic knapsack problem with the following parameters:

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \phi(\mathbf{x}) := 4x_1^2 + \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2 - 2x_2 - 2x_3, \\
\text{subject to} \quad & \sum_{j=1}^{3} g_j(x_j) = x_1 + x_2 + 2x_3 = 4, \\
& 0.5 \leq x_1 \leq 2, \\
& 0.5 \leq x_2 \leq 3, \\
& 0 \leq x_3 \leq 1.
\end{aligned}$$

We relax the lower and upper bounds and search for the optimal solution to the following problem;

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \phi(\mathbf{x}) := 4x_1^2 + \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2 - 2x_2 - 2x_3, \\
\text{subject to} \quad & \sum_{j=1}^{3} g_j(x_j) = x_1 + x_2 + 2x_3 = 4.
\end{aligned}$$

From optimality condition (24a) we compute $\hat{\mathbf{x}}(\mu^1)$;

$$\begin{aligned} x_1 &= -\mu^1/8, \\ x_2 &= 2 - \mu^1, \\ x_3 &= 2 - 2\mu^1. \end{aligned}$$

To find μ^1, we use the resource constraint;

$$\begin{aligned} \sum_{j=1}^{3} g_j(x_j) &= x_1 + x_2 + -2x_3 \\ &= -\mu^1/8 + 2 - \mu^1 + 2(2 - 2\mu^1) \\ &= 6 - \frac{41}{8}\mu^1 = 4 \end{aligned}$$

which gives $\mu^1 = 16/41$. Hence $\hat{\mathbf{x}}(\mu^1) = (-8/41, 65/41, 49/41)$. Now consider the lower and upper bounds, from (25) we have $\mathbf{x}^1 = (0.5, 65/41, 1)$. We evaluate the solution $\mathbf{x}^1$ implicitly;

$$\begin{aligned} \nabla^1 &= 0.5 - (-8/41) = 57/82, \\ \Delta^1 &= 49/41 - 1 = 8/41 \end{aligned}$$

implies that $\Delta^1 < \nabla^1$ and since g_j is increasing and $x(\mu)$ is decreasing in μ we peg $x_1^* = 0.5$. Moreover, we reduce the problem such that

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \phi(\mathbf{x}) := \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2 - 2x_2 - 2x_3, \\ \text{subject to} & \sum_{j=1}^{2} g_j(x_j) = x_2 + 2x_3 = 4 - x_1^* = 3.5. \end{aligned}$$

Still, we have that $x_2 = 2 - \mu^2$ and $x_3 = 2 - 2\mu^2$. Hence,

$$\begin{aligned} \sum_{j=1}^{2} g_j(x_j) &= x_2 + 2x_3 \\ &= 2 - \mu^2 + 2(2 - 2\mu^2) \\ &= 6 - 5\mu^2 = 3.5. \end{aligned}$$

This gives $\mu^2 = 1/2$ which implies that $\hat{\mathbf{x}}(\mu^2) = (1.5, 1)$. Notice that non of the bounds are violated, hence $\hat{\mathbf{x}}(\mu^2) = \mathbf{x}^2 = (1.5, 1)$, $\nabla^k = 0$ and $\Delta^k = 0$. We have $\nabla = \Delta$ which implies that we have found the optimal solution. We summarize and concludes that the optimal solution $\mathbf{x}^* = (0.5, 1.5, 1)$.

2.2.2 Algorithm: Primal determination with Implicit evaluation of the Relaxed problem (PIR)

We assume that Case 1 in Section 2.1.3 holds for problem (1). We post an algorithm for problem (1) which is similar to the algorithms [BrS02, Section 2]:

Step 0: (Check if inequality feasible)
If $\mu = 0$ feasible, set $\mu^* = 0$ and determine $\mathbf{x}^*$ from (13).
Initialization: $\mathbf{J}^k = \{1, 2, ..., n\},\ b^k = b,\ k = 1$
Iterative algorithm:
Step 1: (Solve the relaxed problem)
Find $\hat{x}_j^k$ by solving problem (23).
Step 2: (Calculate reference)
Determine $\mathbf{J}_+^k$ and $\mathbf{J}_-^k$ from (25) while
computing $\Delta^k = \alpha_+^k - \beta_+^k$ and $\nabla^k = \beta_-^k - \alpha_-^k$.
Step 3: (Evaluate)
If $\Delta^k > \nabla^k$, go to Step 4.
If $\Delta^k < \nabla^k$, go to Step 5.
If $\Delta^k = \nabla^k$, Set $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$, $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$
and $x_j^* = x_j^k$ for $j \in \mathbf{J} \setminus \{\mathbf{J}_-^k \cup \mathbf{J}_+^k\}$, stop.
Step 4: (Peg lower bounds)
Set $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$, $b^{k+1} = b^k - \beta_-^k$ and $\mathbf{J}^{k+1} = \mathbf{J}^k \setminus \mathbf{J}_-^k$
If $\mathbf{J}^{k+1} = \emptyset$ then stop, else $k = k + 1$ goto Step 1.
Step 5: (Peg upper bounds)
Set $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$, $b^{k+1} = b^k - \beta_+^k$ and $\mathbf{J}^{k+1} = \mathbf{J}^k \setminus \mathbf{J}_+^k$.
If $\mathbf{J}^{k+1} = \emptyset$ then stop else $k = k + 1$ goto Step 1.

We need to clarify some of the steps in the algorithm. In Step 1, we solve $\hat{\mathbf{x}}^{k+1}$ from, or partly from $\hat{\mathbf{x}}^k$ e.g. assume that $\phi_j = x_j \log(x_j/a_j - 1)$ and $g_j = x_j$, then $x_j^{k+1} = a_j \delta^{k+1}$ where $\delta^{k+1} = b^{k+1} / \sum_{j \in \mathbf{J}^{k+1}} a_j = b^{k+1}/(\omega - \sum_{j \in \mathbf{J}_\pm^k} a_j)$, where $\omega = \sum_{j \in \mathbf{J}^k} a_j$ and we use $\mathbf{J}_+^k$ if the upper bound was pegged in iteration k and $\mathbf{J}_-^k$ if the lower bound was pegged in iteration k. Similar update of $\mathbf{x}_j^{k+1}$ for the quadratic knapsack problem is done in [RJL92, Section 3], [BSS96] and [Kiw08b]. In Step 2, $\alpha_\pm^k = \sum_{j \in \mathbf{J}_\pm^k} g_j(\hat{x}_j)$, $\beta_-^k = \sum_{j \in \mathbf{J}_-^k} g_j(l_j)$ and $\beta_+^k = \sum_{j \in \mathbf{J}_+^k} g_j(u_j)$.

As in [Kiw08b], our algorithm will stop if $\Delta^k = \nabla^k$ while the algorithm in [BrS02, Section 2] stops if $\mathbf{J}_-^k \cup \mathbf{J}_+^k = \emptyset$ but not if $\Delta^k = \nabla^k$. Moreover, in Step 4 and 5, we peg the variables that violate the bounds and calculate b^k explicit while [BrS02] add the index j for the violated bounds to $\mathbf{J}_-$ or $\mathbf{J}_+$

and calculates b^k from $b - \sum_{j \in \mathbf{J}_-} g_j(l_j) - \sum_{j \in \mathbf{J}_+} g_j(u_j)$.

According to Theorem 5 and 6, if Case 2 in Section 2.1.3 holds, Step 3 in the algorithm is modified such that

Step 3': (Evaluate)
If $\Delta^k > \nabla^k$, go to Step 5.
If $\Delta^k < \nabla^k$, go to Step 4.
If $\Delta^k = \nabla^k$, Set $x_j^* = l_j$ for $j \in \mathbf{J}_-^k$, $x_j^* = u_j$ for $j \in \mathbf{J}_+^k$ and $x_j^* = x_j^k$ for $j \in \mathbf{J} \setminus \{\mathbf{J}_-^k \cup \mathbf{J}_+^k\}$, stop.

From Theorem 9, if Case 3 holds for problem (1) then $x_j^* = u_j$ for all $j \in \mathbf{J}$ or if Case 4 holds for problem (1) then $x_j^* = l_j$ for all $j \in \mathbf{J}$.

Remark 2. For the equality problem (2) we allow $\mu < 0$. Hence, the algorithm for problem (2) will be similar to above beside we ignore Step 0.

Remark 3. In Step 1 we have to calculate $\hat{x}_j$ from (24a) $|\mathbf{J}^k|$ times. In Step 2 we need to find x_j from (25) which needs at most $2|\mathbf{J}^k|$ comparisons . We also have to calculate ∇ and Δ which implies $2|\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}|$ operations.

2.2.3 Algorithm: Dual determination with Explicit evaluation of the Relaxed problem (DER)

Stefanov [Ste01] invented an algorithm similar to the algorithm in Section 2.2.2 but instead of evaluate x_j^k from the primal variables (25) he evaluated it from the dual variable (16). The consequence is that we have to calculate the breakpoints $v^l = \left\{\frac{\phi_j'(l_j)}{g_j'(l_j)}\right\}_{j \in \mathbf{J}}$ and $v^u = \left\{\frac{\phi_j'(u_j)}{g_j'(u_j)}\right\}_{j \in \mathbf{J}}$ before starting the algorithm. To evaluate a solution $\mathbf{x}^k$ he uses the explicit evaluation (22a)–(22c) instead of the implicit (29a)–(29c), see Section 2.1.3.

For the algorithm to hold Stefanov shows that the objective functions ϕ_j have to be twice differentiable and convex and g_j has to be twice differentiable, convex, $g_j' > 0$ and of the same form for all $j \in \mathbf{J}$. This is not as general as in [BrS95]. Since Stefanov's modification is equivalent to the reasoning in Section 2.1.3, it can be shown to hold for the same four cases presented in Section 2.1.3.

In the algorithm, we need to calculate the dual variable μ^k of the relaxed problem (23). This is done by substituting $\hat{x}(\mu^k)$ from (24a) into $\sum_{j \in \mathbf{J}^k} g_j(x_j^k) = b^k$.

In the spirit of [Ste01], we modify the algorithm in Section 2.2.2: Assume that Case 1 in Section 2.1.3 holds, if case 2 holds we modify the algorithm similar as in Section 2.2.2. Steps 0, 4 and 5 are similarly to the algorithm

in Section 2.2.2. The other steps and the initialization are modified in the following way:

Initialization: $J = \{1, 2, ..., n\},\ k = 1,\ b^k = b$, calculate v^l and v^u.
Step 1: (Find the dual variable for the relaxed problem)
Find the optimal dual variable μ^k of problem (23).
Step 2: (Calculate reference)
Determine $\mathbf{J}_+^k$ and $\mathbf{J}_-^k$ from (16) and calculate
$g(\mu^k) = \sum_{j \in \mathbf{J}^k \setminus \{\mathbf{J}_+^k \cup \mathbf{J}_-^k\}} g_j(x_j(\mu^k)) + \beta_- + \beta_+$
Step 3: (Evaluate $\delta(\mu^k)$)
If $\delta(\mu^k) = b^k$ stop.
If $\delta(\mu^k) > b^k$ go to Step 4.
If $\delta(\mu^k) < b^k$ go to Step 5.

Notice that with this modification we do not have to calculate $\hat{x}_j^k$ for $j \in \mathbf{J}^k$ in each iteration and we do not have to calculate x_j^k for $j \in \mathbf{J}_-^k \cup \mathbf{J}_+^k$. This might save us some operations and hence cpu-time. However, we have to calculate the optimal dual variable μ^k in each iteration and v^l and v^u once in the initialization.

Remark 4. In Step 1 we need to find μ^k from problem (23). In Step 2 we need at most $2|\mathbf{J}^k|$ comparisons from (25) and we need to calculate x_j^k for $j \in \mathbf{J}^k \setminus \{\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}\}$. For the evaluation we need to calculate $\delta(\mu^k)$ which implies $|\mathbf{J}^k|$ operations.

2.2.4 Algorithm: Dual determination with Implicit evaluation of the Relaxed problem (DIR)

In the previous two sections we have seen two versions of relaxation algorithms. We motivated the advantage of evaluate the dual variable rather than the primal. Now we consider the evaluation of solution x^k for the pegging process. In Section 2.2.3, we use the explicit evaluation (22a)–(22c) of $\mathbf{x}^k$ while in Section 2.2.2, we use implicit evaluation (29a)–(29c) of $\mathbf{x}^k$. Suppose now that we use implicit evaluation for the algorithm in Section 2.2.3, then it is not neccecary to compute x_j^k for $j \in \mathbf{J}^k \setminus \{\mathbf{J}_+^k \cup \mathbf{J}_-^k\}$. This might be a great advantage. Hence, we can modify the initialization and Steps 1 and 2 of the algorithm in Section 2.2.2 such that:

Initialization: $J = \{1, 2, ..., n\},\ k = 1,\ b^k = b$, calculate v^l and v^u.
Step 1: (Find the dual variable for the relaxed problem)
Find the optimal dual variable μ^k of problem (23).
Step 2: (Calculate reference)
Determine $\mathbf{J}_-^k$ and $\mathbf{J}_+^k$ from (16) while computing
$\Delta^k = \alpha_+^k - \beta_+^k$ and $\nabla^k = \beta_-^k - \alpha_-^k$.

Note that we determine $\mathbf{J}_-^k$ and $\mathbf{J}_+^k$ from (16) instead of from (25).

Remark 5. In Step 1 we need to find the optimal dual μ^k solution from problem (23). In Step 2 we need at most $2|\mathbf{J}^k|$ comparisons of (25) but we only need to calculate $\hat{x}_j(\mu)$ for $j \in \{\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}\}$. For the evaluation we need to calculate ∇ and Δ which implies $2|\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}|$ operations.

2.2.5 Algorithm: Dual determination modification with blended evaluation of the Relaxed problem (DBR)

Consider the implicit evaluation; We have to calculate ∇^k and Δ^k from (28a) and (28b) i.e. $\nabla^k = \sum_{j \in \mathbf{J}_-^k} g_j(l_j) - \sum_{j \in \mathbf{J}_-^k} g_j(\hat{x}_j^k)$ and $\Delta^k = \sum_{j \in \mathbf{J}_+^k} g_j(\hat{x}_j^k) - \sum_{j \in \mathbf{J}_-^k} g_j(u_j)$ which implies $2P|\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}|$ operations were P is an integer associated with the number of operation it takes to determine $g_j(x_j)$. Moreover we have to determine $\hat{x}_j$ for $j \in \mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}$ implies $Q|\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}|$ operations were Q is an integer associated with the number of operation it takes to determine $x_j(\mu)$.

Further, consider the explicit evaluation; We have to calculate $\delta(\mu^k) = \sum_{j \in \mathbf{J}^k \setminus \{\mathbf{J}_+^k \cup \mathbf{J}_-^k\}} g_j(x_j(\mu^k)) + \sum_{i \in \mathbf{J}_-^\mathbf{k}} g_j(l_j) + \sum_{i \in \mathbf{J}_+^\mathbf{k}} g_j(u_j)$ which implies $P|\mathbf{J}^k|$ operations. Moreover we have to determine $\hat{x}_j$ for $j \in \mathbf{J}^k \setminus \{\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}\}$ implies $Q|\mathbf{J}^k \setminus \{\mathbf{J}_-^\mathbf{k} \cup \mathbf{J}_+^\mathbf{k}\}|$ operations

Hence, if $(P+Q)|\mathbf{J}^k| < (2P+2Q)|\mathbf{J}_+^k \cup \mathbf{J}_-^k|$ or equivalent $|\mathbf{J}^k| < 2|\mathbf{J}_+^k \cup \mathbf{J}_-^k|$ then there will be less operations to evaluate explicit in Step 2 and it would be more successful to use the algorithm in Section 2.2.3. On the other hand, if $|\mathbf{J}^k| > 2|\mathbf{J}_+^k \cup \mathbf{J}_-^k|$, then there will be less operations if we use the algorithm in Section 2.2.4. So, we propose a new algorithm that evaluate the cardinalities of the sets $\mathbf{J}^k, \mathbf{J}_+^k, \mathbf{J}_-^k$. From the cardinalities we make the decision whether to use explicit or implicit evaluation in Step 2. Consider the following modification of algorithm in Section 2.2.2:

Initialization: $J = \{1, 2, ..., n\},\ k = 1,\ b^k = b$, calculate v^l and v^u.
Step 1: (Find the dual variable for the relaxed problem)
Find the optimal dual variable μ^k of (23).
Step 1.1: (Implicit or explicit)
Determine $\mathbf{J}_+^k$ and $\mathbf{J}_-^k$ from (16).
If $|\mathbf{J}^k| < 2|\mathbf{J}_+^k \cup \mathbf{J}_-^k|$ then continue with algorithm in 2.2.3
else continue with algorithm in 2.2.4.

It is worth noting that both methods in Sections 2.2.4 and 2.2.5 are possible to apply to the four cases of problems in Section 2.2.2.

2.2.6 Convergence of the relaxation algorithms

The proof for convergence of the algorithm in Section 2.2.2 was done by Bretthauer and Shetty in [BrS02, Proposition 3-9] for the inequality problem (1). The proof for the equality problem (2) is similar to the one in [BrS02] besides Theorem 13 (corresponding to Proposition 8 in [BrS02]) and we do not have to prove that $\mu^* \geq 0$.

Let k^* be the iteration where the algorithm terminates. Then the algorithm in Section 2.2.2 (PIR) generates the following optimal solution for problem (2):

$$\mu = \mu^{k^*} = -\phi_j'(x_j^{k^*})/a_j \qquad \forall j \in \mathbf{J}^k \tag{38a}$$
$$\rho_j = \phi_j'(l_j^{k^*}) + \mu^{k^*} a_j \qquad \forall j \in \mathbf{J}^- \tag{38b}$$
$$\rho_j = 0 \qquad \forall j \in \mathbf{J}^{k^*} \cup \mathbf{J}^+ \tag{38c}$$
$$\lambda_j = -\phi_j'(u_j^{k^*}) - \mu^{k^*} a_j \qquad \forall j \in \mathbf{J}^+ \tag{38d}$$
$$\lambda_j = 0 \qquad \forall j \in \mathbf{J}^{k^*} \cup \mathbf{J}^- \tag{38e}$$
$$x_j = l_j \qquad \forall j \in \mathbf{J}^- \tag{38f}$$
$$x_j = u_j \qquad \forall j \in \mathbf{J}^+ \tag{38g}$$
$$x_j = x_j^{k^*} \qquad \forall j \in \mathbf{J}^{k^*} \tag{38h}$$

We prove that the optimal solution generated by PIR (38a)–(38h) satisfies all the KKT-conditions (6a)–(6d) and (7b)–(7d) where (7c) has to be fulfilled with equality. The stop conditions of the pegging algorithm will ensure that condition (7c) is fullfilled which also implies that (6b) is fulfilled. From the evaluation of x_j^k, condition (7d) is satisfied.

Theorem 11. The solution generated by the pegging algorithm (38) satisfies KKT–conditions (6c) and (6d).

Proof. (i) Consider $j \in \mathbf{J}^{k^*}$, this implies $l_j < x_j^{k^*} < u_j$. From (38c) and (38e) we have $\rho_j = \lambda_j = 0$ and it follows that KKT–conditions (6c) and (6d) are satisfied.

(ii) Consider $j \in \mathbf{J}^+$, this implies $\rho_j = 0$ and $x_j^{k^*} = u_j$ from (38g). We have $x_j^{k^*} - u_j = 0$ and from (38c), KKT–conditions (6c) and (6d) are satisfied.

(iii) Consider $j \in \mathbf{J}^-$, proof is similar to (ii). □

Theorem 12. The solution generated by the pegging algorithm (38) satisfies KKT–condition (6a).

Proof. (i) Consider $j \in \mathbf{J}^{k^*}$, we have $\phi_j'(x_j^{k^*}) + \mu^{k^*} a_j - \rho_j + \lambda_j = \phi_j'(x_j^{k^*}) + -a_j \phi_j'(x_j^{k^*})/a_j - 0 + 0 = 0$ from (38a), (38c) and (38e) implies condition (6a) is satified.

(ii) Consider $j \in \mathbf{J}^+$, from (38c) and (38g) and (38d) we have $\phi_j'(x_j^{k^*}) + \mu^{k^*} a_j - 0 + \lambda_j = \phi_j'(u_j) + \mu^{k^*} a_j - \phi_j'(u_j) - \mu^{k^*} a_j = 0$, implies (6a) is satisfied.

(iii) Consider $j \in \mathbf{J}^-$, proof is similar to (ii). □

KKT–conditions (7b) is left to prove. To do this we formulate three lemmata similar to [BrS02, Proposition 5, 6 and 7]:

Lemma 1. For problem (2) evaluated by PIR:

(a) If $\nabla^k > \Delta^k$, then $\mu^{k+1} \geq \mu^k$

(b) If $\nabla^k < \Delta^k$, then $\mu^{k+1} \leq \mu^k$

Proof. We have $a_j > 0$ for all j. First consider (a):

We know $\nabla^k > \Delta^k$ so we want to peg the lower bounds such that $x_j^k = l_j$ for $j \in \mathbf{J}_-^k$. We have

$$\begin{aligned} \sum_{j \in \mathbf{J}^k \setminus \mathbf{J}_-^k} a_j \hat{x}_j^k &= b^k - \sum_{j \in \mathbf{J}_-^k} a_j \hat{x}_j^k, \\ &\geq b^k - \sum_{j \in \mathbf{J}_-^k} a_j l_j^k, \\ &= b^{k+1} = \sum_{j \in \mathbf{J}^{k+1}} a_j x_j^{k+1}. \end{aligned}$$

We also know that $\mathbf{J}^k \setminus \mathbf{J}_-^k = \mathbf{J}^{k+1}$, so we have $\sum_{j \in \mathbf{J}^{k+1}} a_j \hat{x}_j^k \geq \sum_{j \in \mathbf{J}^{k+1}} a_j x_j^{k+1}$. We have that $a_j > 0$ which implies that there exist $j_0 \in \mathbf{J}^{k+1}$ such that $x_{j_0}^k \geq x_{j_0}^{k+1}$. ϕ_j is convex implies that $\hat{x}_j(\mu)$ is decreasing in μ for all j which implies that $\mu(\hat{x}_j) = -\phi_j'(\hat{x}_j)/a_j'$ is decreasing in $\hat{x}_j$ for all j. This implies

$$\mu^k = -\phi_{j_0}'(x_{j_0}^k)/a_{j_0} \leq -\phi_{j_0}'(x_{j_0}^{k+1})/a_{j_0} = \mu^{k+1}. \tag{40}$$

(b) is proved similar as (a).

Lemma 2. For problem (2) evaluated by PIR:

(a) If $\nabla^k > \Delta^k$ and $\nabla^i < \Delta^i$ for $i = k+1, k+2, ..., \gamma-1$ where $\gamma > k+1$, then $\mu^\gamma \geq \mu^k$.

(b) If $\nabla^k < \Delta^k$ and $\nabla^i > \Delta^i$ for $i = k+1, k+2, ..., \gamma-1$ where $\gamma > k+1$, then $\mu^\gamma \leq \mu^k$.

Proof. First we consider (a): We have $\mathbf{J}^k \setminus \mathbf{J}^k_- = \mathbf{J}^{k+1}$ from Theorem 5. Hence, $b^{k+1} = b^k - \sum_{j\in\mathbf{J}^k_-} a_j l_j = \sum_{j\in\mathbf{J}^k\setminus\mathbf{J}^k_-} a_j\hat{x}^k_j + \sum_{j\in\mathbf{J}^k_+} a_j\hat{x}^k_j - \sum_{j\in\mathbf{J}^{k+1}} a_j l_j = \sum_{j\in\mathbf{J}^{k+1}} a_j\hat{x}^k_j - \nabla$, we have

$$b^{k+1} = \sum_{j\in\mathbf{J}^{k+1}} a_j\hat{x}^{k+1}_j = \sum_{j\in\mathbf{J}^{k+1}} a_j x^k_j - \nabla \tag{41}$$

where $\nabla \geq 0$ since $a_j > 0$ and $\hat{x}_j \leq l_j$. This implies that $\sum_{j\in\mathbf{J}^{k+1}} a_j x^{k+1}_j \leq \sum_{j\in\mathbf{J}^{k+1}} a_j x^k_j$. In other words there exist at least one $j = j_0 \in \mathbf{J}^{k+1}$ such that $\hat{x}^{k+1}_{j_0} \leq \hat{x}^k_{j_0}$. We know from Lemma 1 that $\mu^{k+1} \geq \mu^k$ and from that $\mu(x_j)$ is decreasing for all j, this implies that $\hat{x}^{k+1}_j \leq \hat{x}^k_j$ for all $j \in \mathbf{J}^{k+1}$ which implies that $\mathbf{J}^{k+1}_+ \subseteq \mathbf{J}^k_+$. This means that

$$\nabla^k > \Delta^k = \sum_{j\in\mathbf{J}^k_+} (a_j\hat{x}^k_j - a_j u_j) \geq \sum_{j\in\mathbf{J}^{k+1}_+} (a_j\hat{x}^k_j - a_j u_j). \tag{42}$$

Further, for $k+1$ we have $\nabla^{k+1} < \Delta^{k+1}$ which implies $b^{k+2} = b^{k+1} - \sum_{j\in\mathbf{J}^{k+1}_+} a_j u_j = \sum_{j\in\mathbf{J}^{k+1}} a_j\hat{x}^k_j - \nabla - \sum_{j\in\mathbf{J}^k_+} a_j u_j = \sum_{j\in\mathbf{J}^{k+2}} a_j\hat{x}^k_j - \nabla + \sum_{j\in\mathbf{J}^{k+1}_+} (a_j\hat{x}_j - a_j u_j)$. Repeating the same reasoning lead us to

$$b^\gamma = \sum_{j\in\mathbf{J}^\gamma} a_j x^k_j - \nabla^k + \sum_{i=k+1}^{\gamma-1} \sum_{j\in\mathbf{J}^i_+} (a\hat{x}^k_j - a_j u_j) \tag{43}$$

for $\gamma = k+2, k+3, ...$ Now, consider $b^{k+2} = \sum_{j\in\mathbf{J}^{k+2}} a_j\hat{x}^{k+2}_j = \sum_{j\in\mathbf{J}^{k+2}} a_j\hat{x}^k_j - \nabla^k + \sum_{j\in\mathbf{J}^{k+1}_+} (a_j\hat{x}^k_j - a_j u_j)$. It follows from (42) that $\sum_{j\in\mathbf{J}^{k+2}} a_j\hat{x}^{k+2}_j \leq \sum_{j\in\mathbf{J}^{k+2}} a_j\hat{x}^k_j$. Similar to above, there exist a $j_0 \in \mathbf{J}^{k+2}$ such that $\hat{x}^{k+2}_{j_0} \leq \hat{x}^k_{j_0}$. Since $\mu(x_j)$ is decreasing for all j we have that $\mu^{k+2} \geq \mu^k$ and

$$\hat{x}^{k+2}_j \leq \hat{x}^k_j \quad \text{for } j \in \mathbf{J}^{k+2}. \tag{44}$$

Let us now consider $b^3 = \sum_{j\in\mathbf{J}^{k+3}} a_j\hat{x}^{k+3}_j = \sum_{j\in\mathbf{J}^{k+3}} a_j\hat{x}^k_j - \nabla + \sum_{j\in\mathbf{J}^{k+1}_+} (a_j\hat{x}^k_j - a_j u_j) + \sum_{j\in\mathbf{J}^{k+2}_+} (a_j\hat{x}^k_j - a_j u_j)$. From the definition of J^k_+, the conditions

$\nabla^k > \Delta^k$ and $\nabla^i < \Delta^i$ for $i = k+1, k+2, ..., \gamma - 1$ and (44) it is easy to realize that $J_+^{k+2} \cup J_+^{k+1} \subseteq J_+^k$ where $J_+^{k+2} \cap J_+^{k+1} = \emptyset$. Similar to (42),

$$\nabla^k > \Delta^k = \sum_{j \in \mathbf{J}_+^k} (a_j \hat{x}_j^k - a_j u_j) \geq \sum_{j \in \mathbf{J}_+^{k+1} \cup \mathbf{J}_+^{k+2}} (a_j \hat{x}_j^k - a_j u_j),$$

gives that $\sum_{j \in \mathbf{J}^{k+3}} a_j \hat{x}_j^{k+3} \leq \sum_{j \in \mathbf{J}^{k+2}} a_j \hat{x}_j^k$ which again implies that there exists j_0 such that $\hat{x}_{j_0}^{k+3} \leq \hat{x}_{j_0}^k$ which implies that $\mu^{k+3} \geq \mu^k$

If we repeat the same reasoning we can make the conclusion $\mu^\gamma \geq \mu^k$. The proof for (b) is analog to the proof of (a).

□

Lemma 3. For problem (2) evaluated by PIR:
(a) If $\nabla^k > \Delta^k$ then $\mu^{k^*} \geq \mu^k$.
(b) If $\nabla^k < \Delta^k$ then $\mu^{k^*} \leq \mu^k$.

Proof. Follows from Lemmata 1 and 2. □

Theorem 13. For problem (2), the solution generated by PIR (38) satisfies KKT-conditions (7b).

Proof. First we consider the dual variables ρ_j:
(i) For $j \in \mathbf{J}^{k^*} \cup \mathbf{J}^+$, we have from (38c) that $\rho_j = 0$.
(ii) For $j \in \mathbf{J}^-$, we have from (38c) that $\rho_j = \phi_j'(l_j) + \mu^{k^*} a_j$. We know that all $j \in \mathbf{J}^-$ was pegged in the iterations k where $\nabla^k > \Delta^k$. For these iterations k we have $\hat{x}_j^k \leq l_j$. Further, from convexity of ϕ_j, we have that $\mu(x_j) = -\phi_j'(x_j)/a_j$ is decreasing in x_j. Hence

$$\frac{\rho_j}{a_j} = \frac{\phi_j'(l_j)}{a_j} + \mu^{k^*} \geq \frac{\phi_j'(\hat{x}_j^k)}{a_j} + \mu^{k^*} = -\mu^k + \mu^{k^*} \geq 0 \tag{45}$$

since $a_j > 0$. The last inequality follows from lemma 3 (a), $\mu^{k^*} \geq \mu^k$.
(iii) For $j \in \mathbf{J}^{k^*} \cup \mathbf{J}^-$, we have from (38c) that $\lambda_j = 0$.
(iiii) For $j \in \mathbf{J}^+$, we have from (38e) that $\lambda_j = -\phi_j'(u_j) - \mu^{k^*} a_j u_j$. We know that all $j \in \mathbf{J}^+$ was pegged in the iterations where $\nabla^k < \Delta^k$. For these iterations k we have $\hat{x}_j^k \geq u_j$. Further, from convexity of ϕ_j we have that $\mu(x_j) = -\phi_j'(x_j)/a_j$ is decreasing in x_j. Hence

$$\frac{\lambda_j}{a_j} = -\frac{\phi_j'(u_j)}{g_j'(u_j)} - \mu^{k^*} \geq -\frac{\phi_j'(\hat{x}_j^k)}{g_j'(\hat{x}_j^k)} - \mu^{k^*} = \mu^k - \mu^{k^*} \geq 0 \tag{46}$$

since $a_j > 0$. The last inequality follows from lemma 3 (b). □

The algorithms in Sections 2.2.3, 2.2.4 and 2.2.5 will also converge since they are equivalent to PIR.

2.3 Breakpoint algorithms

We present two fundamental breakpoint algorithms. The first one, posted in Section 2.3.2 relaxes the upper bounds and ranks the lower breakpoints. Next it finds an optimal solution to the relaxed problem and then checks if the upper bounds is violated, if that is the case, we peg the variables for the violated upper variables $x_j^* = u_j$ and resolves the relaxed problem. This is done recursively until the upper bounds is not violated. The second class of breakpoint algorithms, described in Sections 2.3.3–2.3.5 finds the optimal solutions from breakpoints at the lower and upper bounds. In Section 2.3.4, we implement mediansearch. In Section 2.3.5, we implement a method that simplifies reduces the comparisons of for determine x_j^k.

2.3.1 The concept of breakpoint algorithms

The idea of breakpoint algorithms is to search for a dual optimal solution among the lower v^l and upper breakpoints v^u. For any given μ, we can determine from (16) if $x_j = l_j$, $x_j = u_j$ or $l_j < x_j < u_j$. This motivate us to define a set of breakpoints:

Definition 14 (Breakpoints)**.**

$$v = v^l \cup v^u \tag{47}$$

Further, define two subsets of v such that $\mathbf{I}_l \cup \mathbf{I}_u = v$ and $\mathbf{I}_l \cap \mathbf{I}_u = \emptyset$. Moreover, let all breakpoints $v_i \in \mathbf{I}_l$ be smaller or equal to the breakpoints $v_i \in \mathbf{I}_u$, that is $v_k \leq v_p$ for all $k \in \mathbf{I}_l$ and for all $p \in \mathbf{I}_u$. Further, define μ_l such that $\mu_l = \max(v_i : i \in \mathbf{I}_l)$ and define μ_r such that $\mu_r = \min(v_i : i \in \mathbf{I}_u)$. Now we can for the following theorem:

Theorem 15. Assume that $g_j(x_j(\mu))$ is nonincreasing in μ. If $\mathbf{I}_l \neq \emptyset$ and $\mathbf{I}_u \neq \emptyset$ is such that $\sum_{j\in\mathbf{J}}^n g_j(x_j(v_l)) > b$ and $\sum_{j\in\mathbf{J}}^n g_j(x_j(v_r)) < b$. Then $v_l < \mu^* < v_r$. If $\mathbf{I}_l = \emptyset$ than $\mu^* \leq v_r$ and if $\mathbf{I}_l = \emptyset$ than $v_l \leq \mu^*$.

Proof. The proof follows from $g_j(x_j(\mu))$ is nonincreasing in μ. □

Note that Theorem 15 is valid for Case 1 and Case 2 in Section 2.1.3. Also note that if $\mathbf{I}_l = \emptyset$ than from (16), we have $x_j^* = u_j$ for all $j \in \mathbf{J}$. Similar if $\mathbf{I}_u = \emptyset$ than $x_j = l_j$ for all $j \in \mathbf{J}$.

Solving problem (1) (assumed that $\mu^* > 0$) or problem (2) boils down to find $\mathbf{I}_l$ and $\mathbf{I}_u$. This can be done by first sorting the breakpoints v and apply an sequential search while evaluating the solution explicitly from (22a)–(22c). Once we found μ_l and μ_r such that Theorem 15 holds, we can find the

dual optimal solution μ^* from (16) and

$$\sum_{j\in\mathbf{J}\setminus\mathbf{J}^-\cup\mathbf{J}^+}^{n} g_j(x_j(\mu)) = b - \sum_{j\in\mathbf{J}^-}^{n} g_j(l_j) - \sum_{j\in\mathbf{J}^+}^{n} g_j(u_j), \tag{48}$$

where $\mathbf{J}^- = \{j : \mu_l \geq v_j^l \text{ for all } j \in \mathbf{J}\}$ and $\mathbf{J}^+ = \{j : \mu_r \leq v_j^r \text{ for all } j \in \mathbf{J}\}$. Several methods for finding μ_l and μ_r of a sorted array od breakpoints have been proposed e.g. sequential search [LuG75], bisection or Fibonacci search [Zip80]. Another common approach for finding the optimal μ_l and μ_r is to evaluate the median v_m of the breakpoints v. From the evaluation of v_m, we can reduce the breakpoints by half and iteratively find the optimal μ_l and μ_r. This method was first applied by Brucker [Bru84].

Example 3. Consider the quadratic knapsack problem with the following parameters:

$$\begin{aligned}
\min_x \quad & \phi(\mathbf{x}) := 4x_1^2 + \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2 - 2x_2 - 2x_3, \\
\text{s.t.} & \sum_{j=1}^{3} g_j(x_j) = x_1 + x_2 + 2x_3 = 4, \\
& 0.5 \leq x_1 \leq 2, \\
& 0.5 \leq x_2 \leq 3, \\
& 0 \leq x_3 \leq 1.
\end{aligned}$$

We can determine the breakpoints for the lower and upper bounds

$$v^l = \left\{ \frac{-\phi_j'(l_j)}{a_j} \right\}_{j=1,2,3} = \{-4, 1.5, 1\},$$

$$v^u = \left\{ \frac{-\phi_j'(u_j)}{a_j} \right\}_{j=1,2,3} = \{-16, -1, 0.5\}.$$

And from these breakpoints we form v_i such that $v_i = \{-16, -4, -1, 0.5, 1, 1.5, 2\}$ as in (47). Now we like to check if $\mu^1 = -16$ is a feasible solution. From (16) we have

$$x_j^1 = \begin{cases} x_j, & \text{if } \mu^1 = -16 = -\phi_j'(x_j)/a_j, \\ l_j, & \text{if } \mu^1 = -16 \geq -\phi_j'(l_j)/a_j, \\ u_j, & \text{if } \mu^1 = -16 \leq -\phi_j'(u_j)/a_j, \end{cases} \tag{50}$$

which implies that $\mathbf{x} = (u_1, u_2, u_3) = (2, 3, 1)$ since $-16 \leq \phi_j'(u_j)/g_j'(u_j)$ for $j = 1, 2, 3$. Now, consider the optimality condition

$$\sum_{j=1}^{3} g_j(x_j) = 2 + 3 + 2 * 1 = 7 \geq 4.$$

Clearly, the solution is not feasible. Hence, we try $\mu^2 = -4$ which implies that $\mathbf{x} = (0.5, 3, 1)$ which gives

$$\sum_{j=1}^{3} g_j(x_j) = 0.5 + 3 + 2 * 1 = 5.5 \geq 4.$$

Next, $\mu^3 = -1$ again gives $\mathbf{x} = (0.5, 3, 1)$. Further, $\mu^4 = 0.5$ gives that $v_2^u < \mu^4 < v_2^l$ which implies that we must compute x_2 from $\mu^4 = 0.5 = -\phi_2'(x_2)/a_2 = (2 - x_2)/1$ implies that $x_2 = 1.5$. As we can see $x_2 = 1.5$ is feasible since $0.5 \leq x_2 \leq 2$. We have $\mathbf{x} = (0.5, 1.5, 1)$, let us see if it is feasible,

$$\sum_{j=1}^{3} g_j(x_j) = 0.5 + 1.5 + 2 * 1 = 4.$$

So it is feasible and optimal since (16) and the feasibility of the resource constraint ensure that all KKT-conditions are satisfied.

If we change the equality constraint $\sum_{j=1}^{3} g_j(x_j) = x_1 + x_2 + 2x_3 = 4$ for a inequality constraint $\sum_{j=1}^{3} g_j(x_j) = x_1 + x_2 + 2x_3 \leq 4$, the optimality condition $\mu \geq 0$ implies that we do not have to consider the negative values of v_i. This can save us much cpu-time.

2.3.2 Algorithm: Relax Upper bounds and Sort lower Breakpoints (RUSB)

Charnes and Cooper [ChC58] developed an algorithm, based on ranking of breakpoints, for a special case of the theory of search problem (67) with a linear resource constraint. The algorithm was extended to a more general problem by Luss and Gupta [LuG75]. Their algorithm concerns problem (2) with ϕ_j strictly convex, decreasing, defined on the positive orthant and $a_j = x_j$ for all $j \in \mathbf{J}$. One should note that Sanathanan [San71] published an article some years earlier with a similar algorithm and Srikantan [Sri63] used a similar method for solving a resource allocation problem. However, Luss' and Gupta's algorithm is divided in two, one inner and one outer algorithm. The inner algorithm solves problem (2) relaxed from the upper bounds, i.e.

$$\underset{x}{\text{minimize}} \quad \phi(\mathbf{x}) := \sum_{j \in \mathbf{N}} \phi_j(x_j), \tag{51a}$$

$$\text{subject to} \quad g(\mathbf{x}) := \sum_{j \in \mathbf{N}} a_j x_j = b^k, \tag{51b}$$

$$l_j \leq x_j, \quad j \in \mathbf{N} = \{1, 2, ..., n\}, \tag{51c}$$

Since Luss' and Gupta's algorithm [LuG75] evaluate the dual variable μ, it is only feasible for Case i in Section 2.5. Zipkin [Zip80] suggest to evaluate the resource constraint (22a)–(22c) instead of the dual variable μ. Then we do not need to determine μ from (48) in each iteration. Hence, we only have to apply a numerical method in the last iteration when $r - l = 1$. We present an algorithm similar to the one in [Zip80] but implemented with bisectional search. The algorithm follows:

Initialization: (Find breakpoints)
Sort $v^N = \{-\phi(l_j)/a_j\}_{j\in\mathbf{N}} \cup \{\pm\infty\}$ such that $v_i^N \leq v_{i+1}^N$ for $i = 1, 2, ..., |\mathbf{N}| + 2$. Set $l = 1, r = |\mathbf{N}| + 2, k = 1, \mathbf{N}' = \mathbf{N}$.
Iterative algorithm:
Step 1: (Stopping test)
If $r - l = 1$ find x_j^k from (16) and (48), stop.
Otherwise $m = \lceil \frac{r+l}{2} \rceil$.
Step 2: (Compute explicite reference)
$\delta = \sum_{j\in\mathbf{N}'} a_j x_j(v_m^N)$ where $x_j(v_m^N)$ is determined from (16).
Step 3: (Optimality check)
If $\delta = b^k$, $\mu^* = v_m^N$ stop.
If $\delta < b^k$ go to Step 4.
Otherwise ($\delta > b^k$) go to Step 5.
Step 4: (Update and fix)
$r = \min_{\arg(i)}\{i : v_i = v_m\}$, $k = k + 1$, go to Step 1.
Step 5: (Update and fix)
If $v_m^N \geq v_j^l$ then $\mathbf{N}' = \mathbf{N}' \setminus \{j\}$, $b^{k+1} = b^k - a_j l_j$, $x_j^* = l_j$.
$l = \max_{\arg(i)}\{i : v_i = v_m\}$, $k = k + 1$, go to Step 1.

The outer algorithm consider the upper bounds, Luss and Gupta referred this algorithm to [Geo70]. Zipkin [Zip80] described this extended algorithm explicitly. We post algorithm (RUSB):

Step 0: (Check if inequality feasible)
If $\mu = 0$ feasible $\mu^* = 0$ and determine $\mathbf{x}^*$ from (13).
Initialization: Set $\mathbf{N} = \{1, 2, ..., n\}$ and $\mathbf{J}^+ = \emptyset$.
Iterative algorithm:
Step 1: (Stopping test)

If $\mathbf{N} = \emptyset$, stop.

Step 2: (Solve relaxed problem (51))
Use the inner algorithm to solve problem (51).

Step 3: (Find offended upper bounds)
Set $\mathbf{J}^+ = \{j : x_j^* > u_j, j \in \mathbf{N}\}$

Step 4: (Optimality check or update)
If $\mathbf{J}^+ = \emptyset$ stop.
Otherwise set $x_j^* = u_j$ for $j \in \mathbf{J}^+$ and set $\mathbf{N} = \mathbf{N} \setminus \mathbf{J}^+$.
Go to Step 1.

Remark 6. The algorithm need to sort an array of n elements (or less if we remove records that are equal). If the problem has upper bounds, we need to resolve the problem, probably several times. This might be time a demanding process.

It is worth noting that Einbu [Ein81] published an extension of Luss' and Gupta's algorithm. He proposed a numerical method for the update of the dual variable. This was also mentioned by Zipkin [Zip80].

2.3.3 Algorithm: Sort Breakpoints (SB)

Year 1969, some years before the publication of Luss' and Gupta's article [LuG75], Bodin [Bod69] had an article published about a ranking method. He treated a separable resource allocation problem with ϕ_j strictly convex, increasing and with the property $\phi_j(l_j) > 0$ for all j and $0 < l_j < u_j < 1$. The problem treated was not as general as Luss' and Gupta's problem but he suggested another method for treatment of the upper bounds. Instead of just rank the lower breakpoints, he also considered the upper breakpoints.

Bodin's algorithms has been generalized and further developed by several authors, e.g. see [FeZ83], [NiZ92], [Bru84], [Kiw07], [HKL80], [KoL98]. Assume that Case 1 in Section 2.2.2 holds for problem (1), we present an algorithm (SB) which is a mixture of the authors mentioned above:

Step 0: (Check feasibility of $\mu = 0$)
If $\mu = 0$ is feasible then $\mu^* = 0$, determine x^* from (13) and stop.

Initialization
Compute v^l and v^u and sort $v^k = \{-\infty, v^l, v^u, \infty\}$ such that

$v_i^k \leq v_{i+1}$ for $i = 1, 2, ..., 2n + 2$.
Set $l = 1$, $r = 2n + 2$, $k = 1$, $b^k = b$, $\mathbf{J} = \{1, 2, ..., n\}$

Iterative algorithm

Step 1: (Stopping test)
If $r - l = 1$, goto Step 6.
Otherwise $m = \lceil \frac{r+l}{2} \rceil$,

Step 2: (Compute explicit reference)
Determine $\delta = \sum_{j \in \mathbf{J}: v_j^u < v_m < v_j^l} g_j(x_j(v_m)) + \beta_+ + \beta_-$ where x_j is determined from (16).

Step 3: (Evaluate)
If $\delta > b^k$, go to Step 4.
If $\delta < b^k$, go to Step 5.
Otherwise ($\delta = b^k$) set $\mu^* = v_m$ and goto Step 6.

Step 4: (Update and fix lower bounds)
For $j \in \mathbf{J}$, if $v_m \geq v_j^l$ then $\mathbf{J} = \mathbf{J} \setminus \{j\}$ and $x_j^* = l_j$.
Set $l = \max_{\arg(i)}\{i : v_i = v_m\}$, $b^{k+1} = b^k - \beta_-$, $k = k + 1$ and go to Step 1.

Step 5: (Update and fix upper bounds)
For $j \in \mathbf{J}$, if $v_m \leq v_j^u$ then $\mathbf{J} = \mathbf{J} \setminus \{j\}$ and $x_j^* = u_j$.
Set $r = \min_{\arg(i)}\{i : v_i = v_m\}$, $b^{k+1} = b^k - \beta_+$, $k = k + 1$ and go to Step 1.

Step 6: (Find the optimal x^*)
Find optimal x^* from (16) and (48).

In Step 2, $\beta_-^k = \sum_{j \in \mathbf{J}: v_j^l \leq v_m} g_j(l_j)$ and $\beta_+^k = \sum_{j \in \mathbf{J}: v_j^u \geq v_m} g_j(u_j)$.

For the Cases 2, 3 and 4 in Section 2.1.3 the algorithm is modified similar as in Section 2.1.3. For problem (2), the algorithm is similar besides Step 0 is vanished.

2.3.4 Algorithm: Median search of Breakpoints (MB)

Since sorting of an array might be a very time demanding task, we may skip the sorting of the breakpoints v for the algorithm in Section 2.3.3. Instead we can apply a median search function in Step 2. This was first done by Brucker [Bru84]. With this modification the algorithm in Section 2.3.3 takes the following form, (Steps 0, 2, 3 and 6 are similar to the algorithm in Section 2.3.3):

Initialization:

Compute v_j^l, v_j^u and let $v^k = \{v^l, v^u\}$, $\mu_l = -\infty$ and $\mu_r = \infty$.
Set $k = 1$, $b^k = b$, $\mathbf{J} = \{1, 2, ..., n\}$

Iterative algorithm:

Step 1: (Stopping test)

If $v^k = \emptyset$, goto Step 6.
Otherwise $v_m = median(v^k)$.

Step 4: (Update and fix lower bounds)

If $v_m \geq v_j^l$ then $\mathbf{J} = \mathbf{J} \setminus \{j\}$ and $x_j^* = l_j$.
Set $v^k = \{v^k : v_m < v^k\}$, $b^k = b^k - \beta_-$, $k = k + 1$ and $\mu_l = v_m$.
Go to Step 1.

Step 5: (Update and fix upper bounds)

If $v_m \leq v_j^u$ then $\mathbf{J} = \mathbf{J} \setminus \{j\}$ and $x_j^* = u_j$.
Set $v^{k+1} = \{v^k : v_m > v^k\}$, $b^k = b^k - \beta_+$, $k = k + 1$ and $\mu_r = v_m$.
Go to Step 1.

In the algorithm, μ_l and μ_r is a lower and an upper bound such that $\mu_l < \mu^* < \mu_r$.

2.3.5 Algorithm: Modified Median search of Breakpoints (MMB)

For the quadratic knapsack problem (63), Kiwiel [Kiw08, Section 3] applies a method for determine if $x_j^k \in [l_j, u_j]$. We generalize Kiwiel's method.

In each iteration the ranking algorithm helps us give a better estimate of the optimal dual variable μ^*. As we have seen this is done by setting v_m to either a lower or an upper bound for the optimal dual variable μ^*. Let us denote the lower bound μ_l and the upper μ_r. Let μ_l and μ_r be such that $\mu_l < \mu^* < \mu_r$. We like state the following theorem:

Theorem 16. Assume that Case 2 holds in Section 2.1.3 i.e. $-\phi_j'(x_j)/g_j'(x_j)$ is decreasing in x_j for all j. If j is such that $\mu_l, \mu_r \in [v_j^u, v_j^l]$ then $l_j < x_j < u_j$.

Proof. Assume that $\mu_l < \mu_r$ and that $\mu_l, \mu_r \in [v_j^u, v_j^l]$ holds for some j. Since $-\phi_j'(x_j)/g_j'(x_j)$ is decreasing in x_j for all j, we have $v_j^u = -\phi_j'(u_j)/g_j'(u_j) \leq \mu_l < \mu^* < \mu_r \leq -\phi_j'(l_j)/g_j(l_j) = v_j^l$. From (16) we have that if $-\phi_j'(u_j)/g_j'(u_j) < \mu^* < -\phi_j'(l_j)/g_j(l_j)$ then $l_j < x_j < u_j$. □

Similar we can prove the following theorem:

Theorem 17. Assume that Case 1 holds in Section 2.1.3. If j is such that $\mu_l, \mu_r \in [v_j^l, v_j^u]$ then $l_j < x_j < u_j$.

From Theorem 16 and 17 it is possible to modify algorithm MB. Let us define a set $\mathbf{M} = \{j : \mu_l, \mu_r \in [v_j^u, v_j^l]\}$. Hence, $j \in \mathbf{M}$ implies that $l_j < x_j^* < u_j$. So if it is determined in iteration k that $j \in \mathbf{M}$ we do not have to use (16) to determine if x_j violate the bounds. This might save us some operations. But we should have in mind that determine $j \in \mathbf{M}$ need some extra operations.

To determine if $j \in \mathbf{M}$ we need some initial values for μ_l^1 and μ_r^1. If $\mu_l^1 = -\infty$ and $\mu_r^1 = \infty$ then $\mu_l^1, \mu_r^1 \notin (v_j^u, v_j^l)$. After the first iteration $\mu_l^2 = \mu_l^1$ or $\mu_r^2 = \mu_r^1$ implies $\mu_l^2, \mu_r^2 \notin (v_j^u, v_j^l)$. So there is no need to check if $j \in \mathbf{M}$ after the first iteration. In fact there is no need to check if $j \in \mathbf{M}$ before iteration k such that $\mu_l^k \geq \mu_l^1$ and $\mu_r^k \leq \mu_r^1$.

We now post an algorithm that applies the use of M where Steps 0, 3 are similar to SB in Section 2.3.3:

Initialization:

Compute v_j^l, v_j^u, let $v^k = \{v^l, v^u\}$, $\mu_l = -\infty$ and $\mu_r = \infty$.
Set $k = 1$, $b^k = b$, $\mathbf{J} = \{1, 2, ..., n\}$ and $\mathbf{M} = \emptyset$.

Iterative algorithm:

Step 1: (Stopping test)
If $v^k = \emptyset$, goto Step 6.
Otherwise $v_m = median(v^k)$.

Step 2: (Compute explicit reference)
Determine $\delta = \sum_{j \in \mathbf{J}: v_j^u < v_m < v_j^l} g_j(x_j(v_m)) + \gamma(v_m) + \beta_+ + \beta_-$ where x_j is determined from (16).

Step 4: (Update and fix lower bounds)
For $j \in \mathbf{J}$: If $v_m \geq v_j^l$ then $\mathbf{J} = \mathbf{J} \setminus \{j\}$ and $x_j^* = l_j$.
If $v_l, v_r \in [v_i^u, v_i^l]$ then $\mathbf{J} = \mathbf{J} \setminus \{i\}$, $\mathbf{M} = \mathbf{M} \cup \{i\}$, update γ.
Set $v^{k+1} = \{v^k : v_m < v^k\}$, $b^{k+1} = b^k - \beta_-$, $k = k + 1$ and $\mu_l = v_m$.
Go to Step 1.

Step 5: (Update and fix upper bounds)
For $j \in \mathbf{J}$: If $v_l \geq v_i^l$ then $\mathbf{J} = \mathbf{J} \setminus \{i\}$ and $x_j^* = u_j$.
If $v_l, v_r \in [v_i^u, v_i^l]$ then $\mathbf{J} = \mathbf{J} \setminus \{i\}$, $\mathbf{M} = \mathbf{M} \cup \{i\}$, update γ.
Set $v^{k+1} = \{v^k : v_m > v^k\}$, $b^{k+1} = b^k - \beta_+$, $k = k + 1$ and $\mu_r = v_m$.
Go to Step 1.

In the algorithm $\gamma(\mu) = \sum_{j \in \mathbf{M}} g_j(x_j(\mu))$ which may be determine partly in Steps 4 and 5 e.g. consider the negative entropy function, $\phi_j = x_j \log(x_j/a_j - 1)$ and $g_j = x_j$, then $\sum_{j \in \mathbf{M}} g_j(x_j^k) = \sum_{j \in \mathbf{M}} a_j e^{-\mu} = de^{-\mu}$ i.e. we update γ in Steps 4 and 5 such that if $v_l, v_r \in [v_i^u, v_i^l]$ then $d = d + c_j$. A similar approach for the quadratic knapsack problem is done by Kiwiel [Kiw08, Section 3].

2.4 Numerical methods

In non-smooth optimization, methods based on the derivative of the objective function $f \in C^0 \setminus C^1$ is not valid. The reason is that the classical gradient is not defined at points $x_0 \in \mathbb{R}$ where $\nabla f(x_0 + \epsilon) \neq \nabla f(x_0 - \epsilon)$ where $\epsilon \in \mathbb{R}$ and $\epsilon \to 0$.

For a function $f \in C^0$ with a gradient $\nabla f(x)$ defined almost everywhere, except for in a finite number or infinite number of points, denote these points as breakpoints. Since the classical derivative is not defined in the breakpoints, another definition is needed. Let $< \cdot, \cdot >$ denote the inner product and define a subdifferiential of f such that:

Definition 18 (Subderivative)**.**

$$\partial f(x) := \{s \in \mathbb{R}^n : f(y) - f(x) \geq < s, y - x > \forall y \in \mathbb{R}^n\}. \tag{52}$$

If the subdifferential f is differentiable at x then $\partial f(x) = \nabla f(x)$ and $\partial f(x)$ is said to be a singleton. If f convex and $0 \in \partial f(x_0)$ than x_0 is a minimum point, [BGLS03, Section 7.1].

Example 4. Consider the L_1-norm $\|x\|$ for $x \in \mathbb{R}^1$ which has a kink in $x = 0$. The related subdifferential due to Definition 18 would be

$$\partial f(x) = \begin{cases} 1, & \text{if } x > 0\ , \\ \text{-1}, & \text{if } x < 0\ , \\ \text{[-1,1]}, & \text{if } x = 0. \end{cases} \tag{53}$$

Note that $0 \in [-1, 1]$, implies an extreme point. Also, note that the subdifferential is identical to the derivative everywhere except for in $x = 0$.

A common stop criterion for smooth optimization is $\|\nabla f(x^k)\| \leq \epsilon$ for $\epsilon \in \mathbb{R}$. This is not always a proper criterion. Consider Example 4 for instance, $s \in \partial f(x)$ is such that $s = \pm 1$ if $x \neq 0$. Assume now that we are using a constant step length. The algorithm will only stop if we choose a starting value such that we end up in the kink.

2.4.1 Zenios' and Nielsen's algorithm (ZN)

Nielsen and Zenios [NiZ92] developed a quasi-Newton method for finding the dual optimal solution μ^* of problem (2). It is assumed that the objective ϕ_j is a strictly convex and separable with a derivative $\phi_j'(x_j)$ which range is $\mathbb{R}$. They compare their numerical method with three breakpoint algorithms and their results shows that their numerical method always performs well compared with the other algorithms. They implement their algorithms

in a massively parallel computer. This is not our intention. But since the algorithm seems to perform well on parallel computers it makes sense to evaluate it on non-parallel computers.

Now, let $f_j(x)$ be the inverse of $\phi'_j(x_j)$ such that:

$$\phi_j(f_j(\mu)) = \mu \quad \forall \mu \in \mathbb{R}. \tag{54}$$

From (25) we can conclude that

$$x_j(\mu) = \max\{l_j, \min\{f_j(a_j\mu), u_j\}\}. \tag{55}$$

The heart of the algorithm is again to find μ such that the primal constraint (1b) is fulfilled, in other words find μ such that

$$\Psi(\mu) = b - \sum_{j=1}^{n} a_j x_j(\mu) = 0. \tag{56}$$

Zenios and Nielsen [NiZ92, Section 1.4] defines two functions Φ_j^+ and Φ_j^+ similar to (55):

$$\Phi_j^+(\mu) = \begin{cases} \min\{f_j(a\mu), u_j\}, & \text{if } a_j > 0, \\ \max\{l_j, f_j(a_j\mu)\}, & \text{if } a_j < 0 \end{cases} \tag{57}$$

and

$$\Phi_j^-(\mu) = \begin{cases} \min\{f_j(a\mu), u_j\}, & \text{if } a_j < 0, \\ \max\{l_j, f_j(a_j\mu)\}, & \text{if } a_j > 0. \end{cases} \tag{58}$$

Note that if $a_j > 0$ and f_j is concave and increasing then Φ_j^+ is concave and if $a_j > 0$ and f_j is decreasing and convex then Φ_j^- is convex. Further, set up two set of indices such that $\mathbf{H}^+ := \{j | a_j > 0\}$ and $\mathbf{H}^- := \{j | a_j < 0\}$. It is now possible to define two approximation of Ψ such that

$$\begin{aligned} \Psi^+(\mu) &= b - \sum_{i=1}^{n} a_j \Phi^+(\mu) \\ &= b - \sum_{j \in \mathbf{H}^+} a_j \min\{g(a\mu), u_j\} - \sum_{j \in \mathbf{H}^-} \max\{l_j, g(a_j\mu)\} \end{aligned} \tag{59a}$$

and

$$\begin{aligned} \Psi^-(\mu) &= b - \sum_{i=1}^{n} a_j \Phi^-(\mu) \\ &= b - \sum_{j \in \mathbf{H}^-} a_j \min\{g(a\mu), u_j\} - \sum_{j \in \mathbf{H}^+} \max\{l_j, g(a_j\mu)\}. \end{aligned} \tag{60a}$$

Now, notice that if $a > 0$ and f_j is concave then is Ψ^+ will be convex and if $a > 0$ and f_j is convex the Ψ^- is concave. Define the sub- and superdifferentials $\Psi^\pm$ for Ψ,

$$\partial\Psi^+(\mu) = \{d \in \mathbb{R} | (\Psi^+(\mu') - \Psi^+(\mu) \geq d(\mu' - \mu) \quad \forall \mu' \in \mathbb{R}\}, \tag{61a}$$
$$\partial\Psi^-(\mu) = \{d \in \mathbb{R} | (\Psi^-(\mu') - \Psi^-(\mu) \leq d(\mu' - \mu) \quad \forall \mu' \in \mathbb{R}\}. \tag{61b}$$

Further, define μ_ϵ^* and $\mathbf{x}_\epsilon^*$ as the approximal dual and primal solution such that $|\Psi(\mu_\epsilon^*)| < \epsilon$ where $\epsilon \in \mathbb{R}$. The algorithm follows ([NiZ92, Linesearch 4]):

Initialization: Set $\epsilon \in \mathbb{R}$, $k = 0$, $\mu^k \in \mathbb{R}$.
Iterative algorithm:
Step 1: (Compute step size).
If $\Psi(\mu^k) > \epsilon$:
$\Delta\mu^{k+1} = -\frac{\Psi(\mu^k)}{d^k}$ where $d^k \in \partial\Psi^+(\mu^k)$, goto Step 2.
else if $\Psi(\mu^k) < -\epsilon$:
$\Delta\mu^{k+1} = -\frac{\Psi(\mu^k)}{d^k}$ where $d^k \in \partial\Psi^-(\mu^k)$, goto Step 2.
else
Set $\mu_\epsilon^* = \mu^k$ and determine $\mathbf{x}_\epsilon^*$ from (16).
Step 2: (Dual variable update)
Set $\mu^{k+1} = \mu^k + \Delta\mu^{k+1}$.
Step 3: $k = k + 1$ Go to Step 1.

The algorithm converges to μ^* such that $\Psi(\mu^*) = 0$ if the objective function components ϕ_j is such that the corresponding function $\Psi^+(\mu)$ is convex or if the corresponding function $\Psi^-(\mu)$ is concave [NiZ92, Proposition 8]. For some problem, the inverse of the derivative might result in imaginary values. One solution of this problem is to consider the equivalent maximization problem of (1) i.e. $\text{maximize}_x -\phi(x)$.

2.4.2 Issues with approximal solutions

If the problem is convex and the optimal dual variable is determined, exactly the primal solution will also be exact. A problem might occur if the dual optimal solution μ_ϵ^* is approximated with a given tolerance $\epsilon > 0$ such that $|\mu_\epsilon^* - \mu^*| = h(\epsilon) = \delta > 0$ for a function h. The map from the dual space to the primal might not be linear. Hence, if x_ϵ^* is the primal optimal solution computed from μ_ϵ^*, then there is no guarantee for the error of x_ϵ^* to be less than δ. The primal error might in fact be much larger than δ such that

$|x^*_\epsilon - x^*| >> \delta$. However, there is methods for generating primal optimal solutions from any Lagrangian dual vector see for example [LMOM07]. Also, we can use μ^*_ϵ as a initial value for a relaxation or breakpoint algorithm.

2.5 Determination of the primal and dual variables

We post three critical cases depending on the character of $\phi_j(x_j)$ and $g_j(x_j)$:

Case i: We can find an explicit closed form of $\hat{x}_j(\mu)$ in (24a). Than it is possible to determine x_j analytically from (25) for all $j \in \mathbf{J}$. Additional, we can find a closed form of μ from

$$\sum_{j \in \mathbf{J}} g_j(\hat{x}_j(\mu)) = b. \tag{62}$$

Case ii: We can find an explicit closed form of $\hat{x}_j(\mu)$ in (24a) and we cannot find a closed form of μ from (62).

Case iii: We cannot find an explicit closed form of $\hat{x}_j(\mu)$ in (24a) and we cannot find a closed form of μ from (62).

If Case ii holds, $\hat{\mathbf{x}}$ has to be determined numerically in Step 1 for the relaxation algorithms in Sections 2.2.2–2.2.5. If Case iii holds, $\hat{\mathbf{x}}$ and x_j has to be determined numerically in Step 1 respectively in Step 2 for the relaxation algorithms in Sections 2.2.2–2.2.5 . For example a Newon method may be applied.

If Case ii holds for the breakpoint algorithms in Sections 2.3, we have to determine μ numerically only one time for determine μ^* in Step 6. If Case iii holds, x_j has to be determined numerically in Step 2.

3 Method for evaluation of algorithms

To achieve a fair comparison between the performance of the algorithms, we need to set up a proper set of test problems. This is done in Section 3.1. In Section 3.2 a brief review of earlier numerical experiments in the same area is presented. With inspiration from these, the properties of the test problem set are set up. Some theory for how the design a problem follows in Section 3.3.

Since it is a diffuse task to decide the goodness of an algorithm, we need a proper measure. In Section 3.4 we give a brief review of Dolan's and Moré's [DoM02] performance profiles. Finally, in Section 3.5, the programming language is presented.

3.1 Problem set

For the numerical study, we consider five common, special cases of problem (2). The reason for evaluating the equality problem (2) is that the algorithm is identically for the inequality problem (1) if $\mu^* > 0$. The case where the optimal solution of problem (1) is such that $\mu^* \leq 0$ or if Cases 3 or 4 in Section 2.1.3 holds, the algorithm is not of interest since it terminates in Step 0. The five problems are briefly specified:

Quadratic problem: The convex separable quadratic problem is described by

$$\phi_j(x_j) = \frac{w_j}{2}x_j^2 - c_j x_j \tag{63}$$

where $w_j, c_j > 0$. Algorithms for problem (63) is widely explored [Kiw07], [Kiw08],[Kiw08b],[NiZ92].

Example 5. Suppose we want to invest an amount of assets, over a fixed period of time, in a given set of n assets. Let x_j be the relative amount invested in asset j. Let r_j denote the expected return of asset j and q_j the covariance of security j. The problem takes the following shape:

$$\begin{aligned} \underset{x}{\text{minimize}} & \sum_{j=1}^{n} \frac{q_j}{2}x_j^2 - \theta r_j x_j, \\ \text{subject to} & \sum x_j = 1, \\ & 0 \leq x_j \leq u_j \end{aligned}$$

where θ is a weight for the importance of the expected value in contrast to the covariance. So, if we want to take a larger risk we increase the value of θ.

This is nothing else than a simplification of the classical portfolio selection problem developed by Markowitz [Mar52]. The theory was simplified by Sharpe [Sha63], he neglected the interrelation between the assets.

Another example of application of problem (63) is minimization of the Euclidean norm.

Stratified sampling If we have a large population and would like to perform a statistical research among the population it is practically infeasible to examine every single individual in the population. Instead we can stratify the population into n strata. Example of a stratum might be people in a certain ages. Let M be the number of individuals in the entire population and M_j the number of individuals in strata j. If we want to minimize the variance of the entire population we need to allocate the number of samples x_j from each strata from:

$$\phi_j(x_j) = \omega_j \frac{(M - x_j)\rho^2}{(M-1)x_j} \tag{65}$$

where $\omega_j = M_j/M$ and σ_j is an estimate of the variance for strata j. From b in the resource constraint we specify the total sample size.

Sampling The stratified sampling problem (65), among other samplings problems can be transformed [BRS99] into

$$\phi_j(x_j) = c_j/x_j. \tag{66}$$

A similar problem is used for determine optimal sample size in air pollution problems; let us assume that we divide an area in n subareas than we consider the objective $\phi(\mathbf{x}) := \sum_{j\in\mathbf{J}} c_j/(x_j - 1)$ where c_j is the related to the emission, [KIM79].

The theory of search In the theory of search problem, we consider how a resource b of time should be spent to find an object with the largest probability. It is assumed that we know the probability m_j for an object to be at area j. The objective ϕ_j describes the probability of finding the object and takes the form:

$$\phi_j(x_j) = m_j(1 - e^{-bx_j}). \tag{67}$$

The problem is possible to apply at a large variation of search problems e.g. searching for refugees fleeing from Cuba [Sto81].

Negative entropy function We also consider the negative entropy function,

$$\phi_j(x_j) = x_j \log(\frac{x_j}{a_j} - 1). \tag{68}$$

3.2 Set up of problem instances

> "However, not testing the algorithm on a large number of functions can easily lead the cynical observer to conclude that the algorithm was tuned to particular functions" [MGB81]

Moré, Garbow and Hillstrom [MGB81] discuss the complications of testing optimization software and stresses the importance of a proper set of problems. They conclude that a large number of test problems are needed, see quotation above.

In [KoL98], Kodialam and Luss solves a problem with lower bounds $l_j = 0$ and no upper bounds. In the numerical experiments, they consider one specific problem of size $n = 10000$ with randomly chose parameters. They divide the set of test problems into 11 groups, considering the percentage of active activities x_j^* for the optimal solution i.e. let $\mathbf{H} = \{j \mid x_j^* > l_j \text{ for } j = 1, 2, ..., n\}$ the percentages is then determined by $|\mathbf{H}|/n$. Several computations of each groups is made and the overall CPU time is determined from the average of these computations. Theirs computational results shows significant differences among the groups.

In [Kiw07], Kiwiel consider 3 different classes of problems; uncorrelated, weakly correlated and strongly correlated parameters belonging to the objective function and the resource constraint for the continuous quadratic problem (63). The results show no obvious differences between uncorrelated and correlated problems. On the other hand, for the integer problem of (1), a significant difference in CPU-time for the correlated and uncorrelated problems is noticed in [MaT90, Chapter 2.10 and 3.5] and [BrS02]. The integer problem is harder to solve when there exist a correlation between the parameters of the objective and the resource constraint, [MaT90]. According to Kiwiel's result it seems not to hold for the continuous quadratic problem (63). Additional, Kiwiel do his numerical experiments for $n = 50,000 - 2,000,000$.

With inspiration and wisdom from the earlier approaches, we divide our set of test problems into groups containing different percentage of the activities within the lower and upper bounds for the optimal solution i.e. let $\mathbf{H} = \{j \in \mathbf{J} \mid l_j \leq x_j^* \leq u_j\}$ the percentage is then determined from $|\mathbf{H}|/n$. We include 6 different numbers n of activities for the problems. Further, in each group 10 random instances are to be solved and the average is

evaluated, see Appendix A. To motivate the set of test problems, we refer to the variance of CPU-time for different percentage of active activities in [KoL98]. Moreover, the theoretical CPU-time for different algorithms vary from $O(n)$ to $O(n^2)$). This motivate the evaluation of different n.

3.3 Design of problems

We need to design problems such that $|\mathbf{H}|/n = y$ where $y \in [0, 1]$. From (16) it is possible to fix an optimal dual variable μ^* such that $|\mathbf{J}^+| = r$, $|\mathbf{J}^-| = p$ and $|\mathbf{J}| = q$, where $r + p = y$ and r, p, q integers such that $r, p, q \geq 0$ and $r + p + q = n$. When μ^* is fixed we need to compute the resource parameter b. This can be done from (16) and the resource constraint $\sum_{j=1}^{n} g_j(x_j^*) = b$.

For the problem set, we randomize the parameters l_j, u_j and all the parameters associated with ϕ_j and g_j. In the numerical study we use a linear resource constraint such that $g_j(x_j) = a_j x_j$, where $a_j > 0$ for $j = \mathbf{J}$. This simplifies the design of the problem set, since ϕ_j is convex and $l_j < u_j$ for all $j \in \mathbf{J}$. Equation (16) becomes

$$x_j^* = \begin{cases} x_j^* & \text{if } \mu^* = -\phi_j'(x_j')/a_j \\ l_j & \text{if } \mu^* \geq -\phi_j'(l_j)/a_j \geq -\phi_j'(u_j)/a_j \\ u_j & \text{if } \mu^* \leq -\phi_j'(u_j)/a_j) \leq -\phi_j'(l_j)/a_j \end{cases} \tag{69}$$

By using the properties of (69), we can determine r, p and q such that $|\mathbf{H}|/n = y$ for any $y \in [0, 1]$. This is done by randomizing l_j, u_j, a_j and the parameters associated with ϕ_j, determine v^l and v^u, then fix μ^*, find $\mathbf{x}^*$ from (16) and determine b from the resource constraint fulfilled with equality.

3.4 Performance profiles

Dolan and Moré [DoM02] discuss the complications that occur, associated with evaluating optimization software. Because of the issues they propose a performance profile for evaluation of optimization software.

We want to evaluate a set of algorithms $\mathbf{A}$ that consists of n_a various algorithms. For the evaluation we have a problem set $\mathbf{P}$ that consists of n_p problem instances. The method Dolan and Moré propose is as follows: We define a set of cpu-times $t_{p,a}$, were $t_{p,a}$ is equal to the time it takes for algorithm a to solve problem p. A performance ratio is introduced:

$$r_{p,a}(t_{p,a}) = \frac{t_{p,a}}{\min\{t_{p,l} : l \in \mathbf{A}\}}. \tag{70}$$

For a problem p, the performance ratio is a measure for how fast algorithm a is relative to the fastest algorithm solving problem p. Fix a constant r_M

such that $r_M \geq r_{p,a}$ for all $p \in \mathbf{P}$, $a \in \mathbf{A}$ and let $r_{p,a} = r_M$ if algorithm a fails to solve problem p. Further, we introduce the distribution ρ_a

$$\rho_a(\tau) = \frac{1}{n_p}|\{p \in \mathbf{P} : r_{p,a} \leq \tau\}|, \tag{71}$$

for each algorithm, where $|\cdot|$ denote the cardinality of a set and $\tau \in [1, r_M]$. For algorithm a, the distribution ρ_a describes the percentage of problem instances that are solved faster or equal fast as τ times the fastest algorithm for problem p. The distribution $\rho(\tau)_a$ has some nice properties: First, $\rho_a(1)$ is the percentage for algorithm a being fastest. $\rho_a(1) < \rho_{a'}(1)$ does not guarantee that algorithm a is better than algorithm a' since algorithm a might be a horrible for all the problems where it was not fastest. Second, $\lim_{\tau \to r_M} \rho_s(\tau)$ is the probability that algorithm a will solve a problem in $\mathbf{P}$. Third, if we have a large problem set $\mathbf{P}$ then $\rho_a(\tau)$ will not be affected much by a small change in $\mathbf{P}$ [DoM02, Theorem 1]).

3.5 Program language, computer and code

The algorithms are implemented in Fortran 95 compiling with *gfortran* under UNIX.

Sorting and searching For the sorting procedure in the breakpoint algorithms we use quicksort implemented similarly to the one in [Knu98, Section 5.2.2]. Quicksort has a worst-case running time of $O(n^2)$ and an average $O(n\log(n))$ where the constants in $O(n\log(n))$ are small ([CLRS09]). For large n, quicksort is often the best choice in practice ([CLRS09], [PTVF92, Section 8.2]).

For the median search method we use an algorithm that operates similarly to quicksort. We implement the search method similarly as in [PTVF92, Section 8.5]. The algorithm has a worst-case running time of $O(n)$.

4 Computational experiments

We show the results from numerical experiments of the problems in Section 3.1. The problem set is evaluated for $n = 50,000 - 2,000,000$.

For the theory of search problem (67), the parameters was randomized in the following intervals: $m_j \in [0.5 - 8]$, $b_j \in [0.1, 3]$, $a_j \in [1, 3]$, $l_j \in [0, 0.1]$ and $u_j \in (0.1, 5]$.

For the quadratic problem (63), the parameters was randomized such that $a_j \in [1, 30]$, $w_j \in [1, 20]$, $c_j \in [1, 25]$, $l_j \in [0, 3]$ and $u_j \in (3, 11]$.

For the stratified sampling problem (65), the parameters was randomized such that $a_j \in [1, 30]$, $m_j \in [5, 30]$, $c_j \in [1, 4]$, $l_j \in [1, 3]$ and $u_j \in (3, 15]$.

For the negative entropy problem (68), the parameters was randomized such that $c_j \in [50, 250]$, $l_j \in [20, 100]$ and $u_j \in (30, 210]$.

For the production problem (66), the parameters was randomized such that $a_j \in [1, 4]$, $c_j \in [5, 30]$, $l_j \in [0, 3]$ and $u_j \in (3, 6]$.

In Section 4.1 we present performance profiles for the relaxation algorithms described in Section 2.2, in Section 4.2 we present the performance profiles for the ranking algorithms described in Section 2.3. In Section 4.3, the best performing relaxation algorithm and breakpoint algorithm are compared with the numerical algorithm in Section 2.4.

4.1 Relaxation methods

The four algorithms in 2.2 are compared successively. All the four algorithms are programmed uniformly i.e. the only things that differs the algorithms is the evaluation of the dual or primal variables and the stop criterion.

4.1.1 PIR vs DIR

We compare algorithm (PIR) in Section 2.2.2 and algorithm (DIR) in Section 2.2.4. Remember, PIR determines the primal variable and evaluates the solution implicitly while DIR determines the dual variable and evaluates the solution explicitly.

As we can see in Figure 1, DIR is fastest or equally fast in 91.7% of the solved problems, while PIR is faster in 8.3% of the solved problems. This is due to the negative entropy problem (68), see Figure 2.

This result is what we can expect from the theory since DIR does not need as many operations in Step 1, see Sections 2.2.2 and 2.2.4. The reason for PIR to be faster in many cases for the negative entropy problem is due to the easy expression of the primal variables $x_j^k(\mu^k) = \frac{a_j}{\sum_{j\in\mathbf{J}^k} c_j}$ while the dual variable is evaluated from $\mu^k = \log \sum_{j\in\mathbf{J}^k} c_j - \log b$. But still, DIR is faster in most of the solved negative entropy problems, see Figure 2.

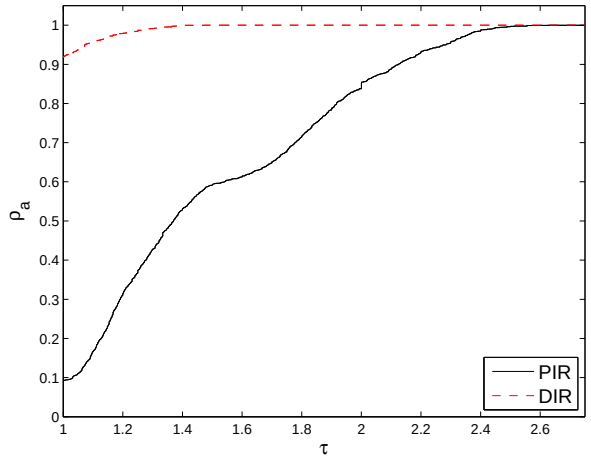

Figure 1: Performance profiles for PIR and DIR. The numerical experiment was done according to Section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000; 2,000,000$ and for the problems in Section 3.1. The algorithms were implemented uniformly.

Summary: For the problem set, it is more profitable to evaluate the dual variable even if we have to compute all the breakpoints in the beginning of the algorithm. Hence, DIR seems to outperform BI.

4.1.2 Explicit vs implicit evaluation

We consider the performance profiles of the CPU-times for the three relaxation algorithms described in Sections 2.2.3 (DER), 2.2.4 (DIR) and 2.2.5 (DBR). The results are in favour of DIR and DBR, see Figure 3. DIR is fastest in 49.1%, DBR is fastest in 56.9% and DER is fastest in 1.4% of the problems solved.

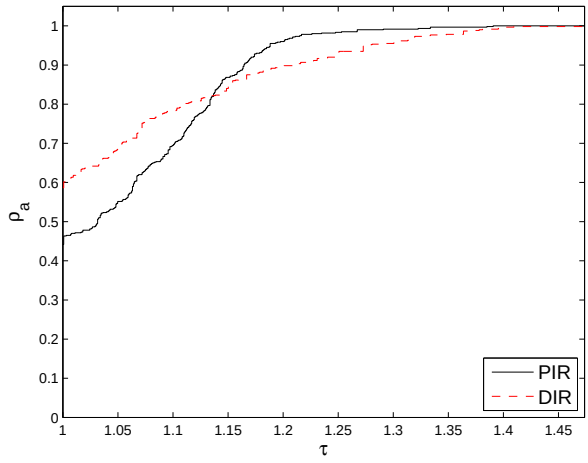

Figure 2: Performance profiles for PIR and dual DIR. The numerical experiment was done according to Section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000; 2,000,000$ for the negative entropy problem (68). The algorithms were implemented uniformly.

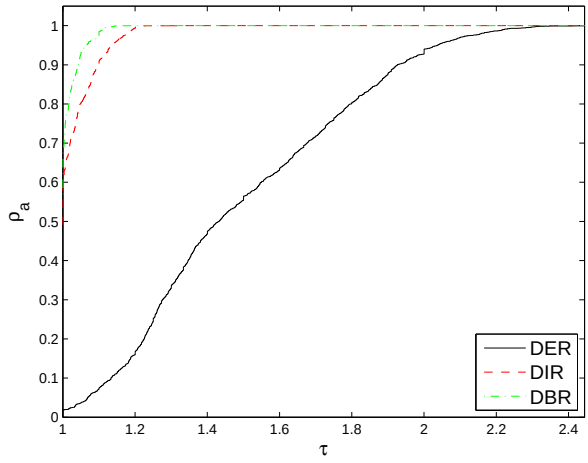

Figure 3: Performance profile for DER, DIR and DBR. The numerical experiment was done according to Section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000; 2,000,000$ for the problems in Section 3.1. The algorithms were implemented uniformly.

Figure 3 shows that the performance of DIR and DBR is very similar. DIR is more often faster than DBR. DIR is never more than 1.20 times

slower than the fastest algorithm while DBR is never 1.10 times slower than the best performing algorithm. As we can see, in average DBR perform just a little better than DIR.

Summary: For the problem set, the results shows that in most cases it more profitable to evaluate a solution $\mathbf{x}^k$ of iterate k implicitly from (29a)–(29c) rather than explicit from (22a)–(22c).

The small difference between the performance profiles in Figure 3, implies that in most cases $|\mathbf{J}^k| > 2|\mathbf{J}_-^k \cup \mathbf{J}_+^k|$. We conclude that DBR performs slightly better than DIR. This agrees with the theory in Section 2.2.5.

4.2 Breakpoint methods

Step by step the performance of the algorithms in Section 2.3 are compared. Since the algorithms are programmed uniformly we eliminate the algorithms that perform just a little bit worse than another. But one should note that this might hold only for the problems in our problem set.

4.2.1 RUSB vs SB

In Sections 2.3.2 and 2.3.3 we showed two fundamental breakpoint algorithms for solving problem (1), RUSB that relaxing the upper bounds and sorting a sequence of size n and SB that sorting a sequence of $2n$ breakpoints. We evaluated these two algorithms for the problem set in Section 3.1 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000$. The results show a significant advantage for SB since RUSB is, on average, 2.7 times as slow and more than 4 times as slow in 10% of the cases, see Figure 4.

RUSB is faster in 2.8% of the cases and this is especially when none of the optimal variables x_j^* violate the upper bounds, i.e. $x_j < u_j$ for all $j \in \mathbf{J}$. However, SB is never more than 30% slower than RUSB.

Summary: SB is generally much faster than RUSB. The reason is that RUSB have to resolve the problem several times. Hence, it more profitable to sort an array of size $2n$ and solve the problem one time than sort a sequence of size n and resolve the problem relaxed from the upper bounds several times. It is expected that RUSB should be faster in the cases where $\mathbf{J}^+ = \emptyset$ since the algorithm do not resolve the relaxed problem.

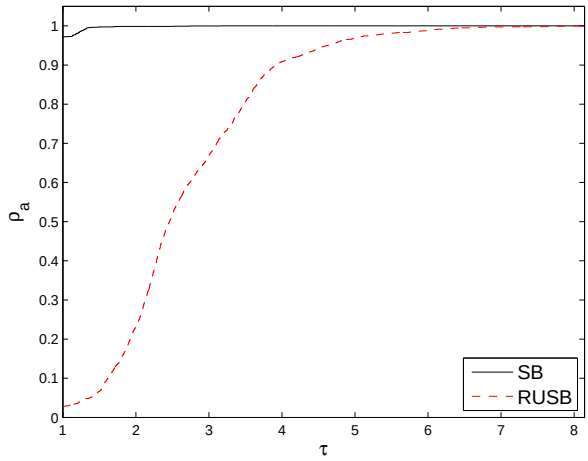

Figure 4: Performance profile for SB and RUSB. The numerical experiment was done according to section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000$ and for the problems in Section 3.1. On average, the cpu-time for RUSB is 2.7 times slower then SB. The algorithms were implemented uniformly.

4.2.2 Sorting vs Median search

In Section 2.3 we presented two different versions of a bisection search. The first method was to sort the sequence of breakpoints v_i which needs to be done only once. The second method was to find the median of v_i which requires to be done $\log_2 n$ times for a reduced sequence of size $n/\lfloor 2^k \rfloor$ for $k = 1, 2, .., \lfloor \log_2 n \rfloor$.

In Figure 5 the performance profile for sorting an array of size n and the median function are shown. The CPU-time for the median function was evaluated for $\log_2 n$ arrays of decreasing order. As we can see, it is approximated 2.7 times faster to use median search.

To ensure that the algorithms are implemented such that median search is faster than quick sort, we present the performance profile for one ranking algorithm using sort (SB) and one using median search (MB), see Figure 6. The performance profile shows that median search is to prefer in front of sorting. The CPU-time differs approximately from being 1.2 to 2.0 times as large in favour for the median search. This is a decrease compared with Figure 5. The reason is obvious: the sorting/median search is weighted with the other part of the algorithms.

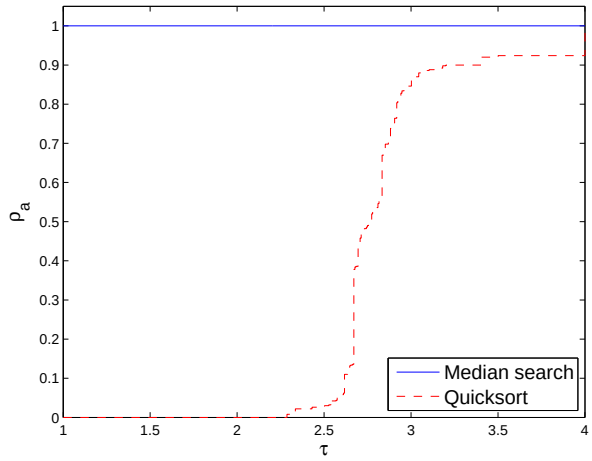

Figure 5: Performance profile for median search and quicksort. The sorting was done only once, while the median search was done $\log_2 n$ times where the array was reduced by half in each iteration. In numerical experiments 1200 randomized arrays of size $n = 100,000 - 8,000,000$ was used.

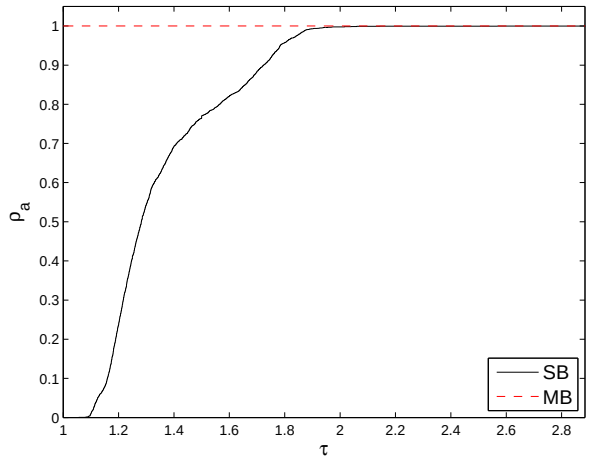

Figure 6: Performance profile for MB and SB. The numerical experiment was done according to Section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000; 2,000,000$ and for the problems in Section 3.1. The algorithms were implemented uniformly, i.e. the only thing that separated the algorithms was the method for finding the median.

Summary: We should prefer median search in front of sorting since it performs better for all the solved problems. In [Kiw08], Kiwiel compare a ranking algorithm in the spirit of MB, Brucker's classic algorithm [Bru84], a modification of Brucker's algorithm, a ranking algorithm with approximal medians and the ranking algorithm in [CaM87]. He consider the the quadratic problem (63) and $n = 50,000 - 2,000,000$. Kiwiel's results shows that the ranking algorithm in the spirit of MB perform better than all the other algorithms.

4.2.3 MBvsMMB

We compare MB as in Section 2.3.3 with MMB as in Section 2.3.5. Remember, MMB evaluates if $x_j^k \in (l_j, u_j)$. Figure 7 shows that MMB is faster in 99% of the cases. We can see that in the few cases where MB is faster, MMB is almost as fast. From the results it seems like it repay to determine M. This is not very strange since when determining M, we do not need to determine x_j from (16) for $j \in$ M and we do not have to compute $\sum_{j \in \mathrm{M}} a_j x_j(\mu^k)$ in each iteration.

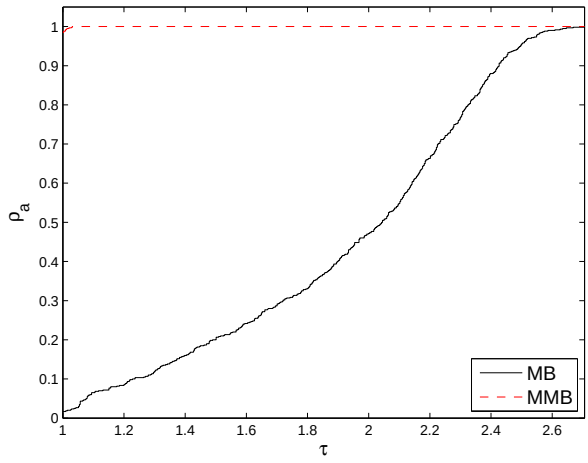

Figure 7: Performance profile for MB and MMB. The numerical experiment was done according to Section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000; 2,000,000$ and for the problems in Section 3.1. The algorithms were implemented uniformly.

4.3 A final comparison between breakpoint, quasi-Newton and relaxation methods

In this section we compare the best performing relaxation algorithm DBR, the best performing breakpoint algorithm MMB and the numerical Newton method ZN in Section 2.4.1. For the numerical method we use the stopping criteria $\mid \sum_{j\in\mathbf{J}} a_j x_j - b \mid / \mid b \mid < 0.1/n$. This will of course give us an approximal optimal solution. The initial value is set to $\mu^0 = (\sum_{j\in\mathbf{J}} \phi_j(l_j)/a_j + \sum_{j\in\mathbf{J}} \phi_j(u_j)/a_j)/(2n)$. If the algorithm does not converge within 100 seconds, we start over and tries $\mu^0 = (\sum_{j\in\mathbf{J}} \phi_j(l_j)/a_j)/n$ and $\mu^0 = (\sum_{j\in\mathbf{J}} \phi_j(u_j)/a_j)/n$. If the algorithm is not solved within 300 seconds we break the algorithm and consider the problem as unsolved.

Figure 8 shows the performance profile for the algorithms. In general, we can see a significant advantage of using the DBR since it is fastest for 95.0% of the solved problems. Moreover, DBR is never more than 1.25 times as fast as the best performing algorithm, which is quite impressive if we compare with the cases when ZN is starting with an initial value μ^0 close the approximated optimal solution μ^*_ϵ.

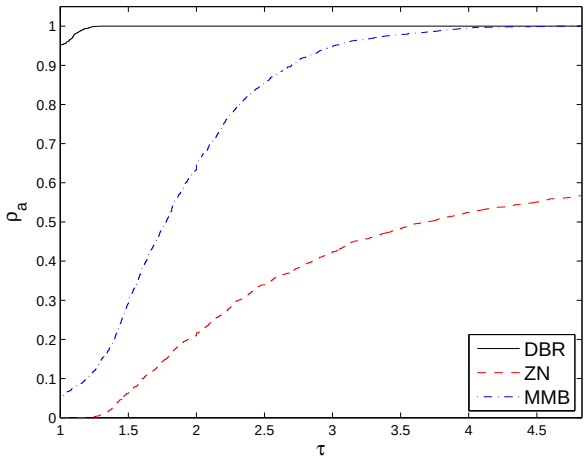

Figure 8: Performance profile for DBR, MMB and ZE. The numerical experiment was done according to Section 3.2 for $n = 50,000; 100,000; 200,000; 500,000; 1,000,000; 2,000,000$ for the problems in Section 3.1.

The breakpoint algorithm MMB, is fastest in 2.4% of the solved problems. This is due to the theory of search problem (67) where MMB is fastest in 25.0% when $|\mathbf{H}|/n < 0.3$. MMB was never faster than DBR for prob-

lems (63),(65), (66) and (68). MMB can compete with DBR when $|(\mathbf{H})|/n$ is small for problem (67), see Appendix A.

For the solved problems, ZN is never fastest. ZN performs well for the theory of search problem (67) and the stratified sampling problem (65) when $|\mathbf{H}|/n > 0.8$. ZN was not very successful for the quadratic problem (63), the negative entropy problem (68) and the sampling problem (66). We can see from the performance profile that in 50% of the cases, ZN is more than 4 times slower than the fastest algorithm. In 6.2% $(186/3000)$ of the cases the algorithm did not solved the problem and this was mostly due to the stratified sampling problem (65) when $|\mathbf{H}|/n < 0.3$, but also for some logarithmic (68) problems when $|\mathbf{H}|/n < 0.3$, see Appendix A.

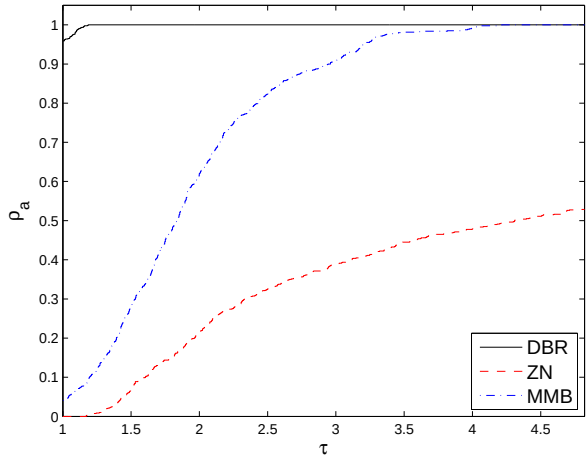

Figure 9: Performance profile for DBR, MMB and ZE. The numerical experiment was done according to Section 3.2 but with 2 problem instances for each group and for $n = 4,000,000; 6,000,000; 8,000,000; 10,000,000; 15,000,000; 20,000,000; 25,000,000; 30,000,000$ for the problems in Section 3.1.

Comparing Figure 8 and 9, we can see that they are very similar. In other words, relative to each other, the performance of the algorithms seems to be almost constant for different sizes of problems n.

For the MMB, the result shows that the cpu-time decreases when $|\mathbf{H}|/n$ increases for the problems in the problem set. The difference in cpu-time for solutions belonging to $|\mathbf{H}|/n > 0.90$ is approximal 2.2 times as long as for solutions belonging to $|\mathbf{H}|/n < 0.10$. For the DBR, the result shows no clear dependent of $|\mathbf{H}|/n$. But we note that, the fastest solution group and the slowest solution group, differ in approximal 10%.

Concerning the numerical method ZN it the opposite holds, i.e. the cpu-time increases when $|\mathbf{H}|/n$ increases, see Appendix A.

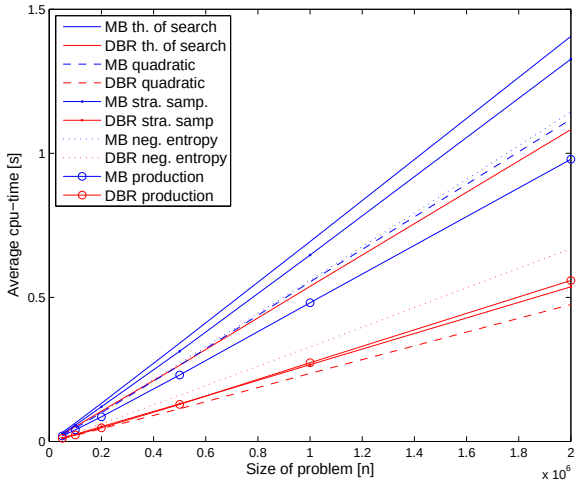

Figure 10: The average cpu-time is plotted as a function of the size of the problem n for DBR and MMB.

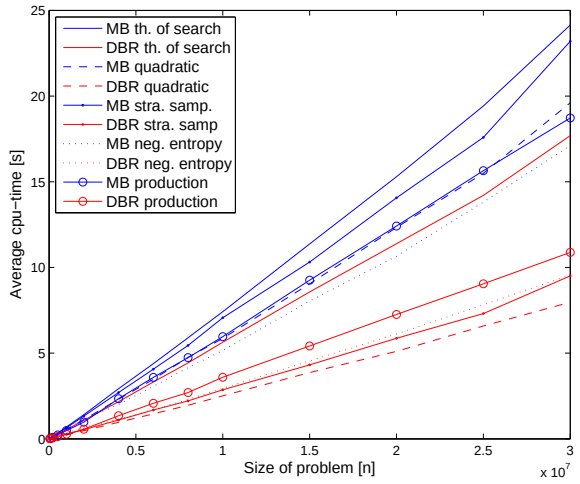

Figure 11: The average cpu-time is plotted as a function of the size of the problem n for DBR and MMB.

In Figure 10 and 11, we can see that the CPU-time for MMB and DBR is as good as linear for $n \in [50,000; 2,000,000]$ respectively $n \in [50,000;$

$30,000,000]$. This is impressive since MMB have a theoretical time complexity of $O(n\log(n))$ and DBR have a time complexity of $O(n^2)$. Also note that all problems, besides the theory of search problem (67), is solved almost equally fast.

Summary: Even if the stopping criteria for ZN is quite loose ($\mid \sum_{j\in\mathbf{J}} a_j x_j - b \mid / < \epsilon$ where $\epsilon \approx 10^-4$), ZN is outperformed by DBR and MMB. Hence, we see no reason to evaluate stronger stopping criteria for ZN nor evaluate methods for finding the optimal solution $\mathbf{x}^*$ from the approximated solution $\mathbf{x}^*_\epsilon$ generated by ZN. When $|\mathbf{H}|/n$ is small, a large part of the x^*_j equals the lower and upper bounds which implies that ZN has to operate in an area where the gradient of (60) and (61) is non-continuous in many directions. Clearly this area is difficult for ZN in a numerical sense, see the cases when $|\mathbf{H}|/n$ in Appendix A. Also in many cases, ZN fails to solve the stratified sampling problem (65). This is probably due to the stiffness of the problem. We should have in mind that ZN just generate an approximal solution. Hence, to be fair against MMB and DBR, we like to note that we need some additive method for ZN to find the optimal solution. But again, since ZN do not perform very well, not even for a loose stop criteria we do not investigate such a method.

Opposite to ZN, MMB perform very good for problems where $|\mathbf{H}|/n$ is small. This is much due to the pegging process, since we peg a large part of $x^k_j \in \mathbf{J}^k$ in each iteration, the problem is reduced in next iteration. This follows from the reduction of breakpoints by half in each iteration. Hence, in contrast to the relaxation algorithm we can guarantee that a large part $x^k_j \in \mathbf{J}^k$ is pegged in each iteration. This might be the reason for MMB to be faster than DBR in some of the cases when $|\mathbf{H}|/n$ is small. However, DBR performs best in most of the cases. It is worth noting that DBR never performs poor, see Appendix A.

Agreement with earlier approaches: In [Kiw08b], Kiwiel compare a relaxation algorithm similar to PIR with a ranking algorithm similar to MMB. He consider the quadratic problem (63) for $n = 50,000 - 2,000,000$. The results show that the relaxation algorithm is slightly faster. Similar results is shown in [RJL92], but for $n = 100 - 4,000$. The results in [Kiw08b] and [RJL92] agrees with our results if we compare the the performance PIR relative to DIR in Figure 3 and the MMB relative to DBR in Figure 8

Kodialam and Luss [KoL98] compare a relaxation algorithm, a breakpoint algorithm using quicksort and a hybrid of these two. In their numerical experiments they consider 110 instances of a problem with an exponen-

tial objective function with a non-linear resource constraint for $n = 10,000$. They conclude that the relaxation algorithm performs best. Hence, their result is coherent with ours.

Zenios and Nielsen [NiZ92] compare their numerical method ZN with a breakpoint algorithm (using quicksort and no pegging) for the quadratic problem. They implement the algorithms on massively parallel computers. Five classes of the quadratic problem are solved. In the results it is shown that ZN performs better than the breakpoint algorithm in 4 of 5 classes problems. In [NiZ92], the over all stop criterion is set to $\mid \sum_{j \in \mathbf{J}} a_j x_j - b \mid < 10^-4$ which is similar to our stop criterion. Clearly, ou results do not agree with the results in [NiZ92]. But we like to stress that there are two important differences in our numerical experiments compared to Zenios' and Nielsen's; We do not use parallel computers and we use a more effective breakpoint algorithm. Moreover, it's not clear what kind of initial values Zenios and Nielsen use.

It is worth noting that Bitran and Hax [BiH81] compared their algorithm similar to BH with Luss' and Gupta's algorithm [LuG75] similar to MMB but with a sequencial search instead of median search and without pegging. In their numerical experiments they concluded that the CPU-time for the relaxation algorithm is comparable by the time for compute the derivatives and sort them, which is done in the initialization of Luss' and Gupta's algorithm. However, they considered a set of test problems similar to our test set but for n between 50 and 200. After many years development of the breakpoint algorithm we can still see it outperformed by the relaxation algorithm. But the differences in CPU-time is not so large anymore.

4.4 Future work

For the numerical experiments in this study, we have assumed that we can find a closed form of the dual variable $\mu(\mathbf{x})$. If that is not the case we need to add a numerical method for finding μ^k in each iteration for the relaxation algorithms. We note that breakpoint algorithm MMB, do not need a closed form of μ besides in the last iteration. However, in [KoL98], a primal breakpoint algorithm, a dual breakpoint algorithm and a relaxation algorithm are compared for a non-linear problem of size $n = 10,000$. The relaxation algorithm and the dual breakpoint algorithm uses a numerical method to determine μ^k in each iteration. In their numerical experiment the primal breakpoint algorithm performs best. As far as we know, this is the only approach that has been done for problems where μ^k does not have a closed form. Hence, an intersting future work is to evaluate problems

where μ does not have a closed form where also a Newton-like method is considered.

4.5 Conclusion

Our results, as well as the results in [Kiw08b],[RJL92] and [KoL98], implies that we should prefer to use a relaxation algorithm when we can find a closed form of $\mu(\mathbf{x})$.

What distinguish the results in our study from earlier is that we showed that our new implementations (DIR and DBR) of the relaxation algorithm outperforms the earlier used algorithms (PIR and DER). In other words, the results shows that we should evaluate the dual variable for the relaxation algorithms i.e. DIR outperforms PIR. Moreover it is more profitable in theory, as well as in practice, to apply blended evaluation i.e. DBR outperform DER and DIR. We also showed that both DBR and MMB perform better than the Newton-like algorithm ZN. Finally, according to the results, MMB and DBR have a time complexity of $O(n)$ for large n in practise.

References

[AEP05] N. Andréasson, A. Evgrafov, and M. Patriksson, *An Introduction to Continuous Optimization*, Studentlitteratur, Lund, 2005.

[BGLS03] J.F. Bonnas, J.C. Gilbert, C. Lemaréchal and C.A. Sagastizábal, *Numberical Optimization; Theoretical and Practical Aspects*, Springer-Verlag, Berlin Heidelberg, 2003.

[BiH81] G. R. Bitran and A. C Hax, *Disaggregation and resource-allocation using convex knapsack-problems with bounded variables*, Management Science, 27 (1981), pp. 431-441.

[BiT89] G. R. Bitran and D. Tirupati, *Tradeoff curves, targeting and balancing in manufactoring networks with queueing networks*, Operational Research, 37 (1989), pp. 547-564.

[Bod69] L. Bodin, *Optimization procedures for the analysis of coherent structures,* IEEE Transactions on Reliability, R-18 (1969), pp. 118-126.

[BrS02] K. M. Bretthauer and B. Shetty, *A pegging algorithm for the nonlinear resource allocation problem,* Computers & Operations Research, 29 (2002), pp. 505-527.

[BrS95] K. M. Bretthauer and B. Shetty, *The nonlinear resource allocation problem, Operations Research,* 43 (1995), pp. 670-683.

[BRS99] K. M. Bretthauer, A. Ross, and B. Shetty, *Nonlinear integer programming for optimal allocation in stratified sampling,* European Journal of Operational Research, 116 (1999), pp. 667-680.

[BSS96] K. M. Bretthauer and B. Shetty, A. Syam *A projection method for integer quadratic knapsack promblem*. Journal of Operational Research, 47 (1996), pp. 457-462.

[Bru84] P. Brucker,*An O(n) algorithm for quadratic knapsack problems,* Operations Research Letters, 3 (1984), pp. 163-166.

[CAA57] C. W. Churchman, R. L. Ackoff, and E. L. Arnoff, *Introduction to Operations Research,* John Wiley & Sons, New York, NY, first ed., 1957.

[CaM87] P. H. Calamai and J. J. Moré, *Quasi-Newton updates with bounds,* SIAM Journal on Numerical Analysis 24 (1987) pp. 1434-1441.

[ChC58] A. Charnes and W. W. Cooper, *The theory of search: Optimal distribution of search effort,*Management Science, 5 (1958), pp. 44-49.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *Introduction to Algorithms,* 3rd ed. The MIT press (2009) Massachusetts Institute of Thechnology

[DoM02] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles,* Math. Program., Ser A 91 (2002) pp. 201-213.

[DSV07] K. Dahiya, S. K. Suneja and V. Verma, *Convex programming with a single separable constraint and bounded variables,* Comput Optim. Applic, 36 (2007) pp. 67-82.

[Ein81] J. M. Einbu, *Extension of the Luss-Gupta resource allocation algorithm by means of first order approximation techniques, Operations Research,* 29 (1981), pp. 621-626.

[FeZ83] A. Federgruen and P. Zipkin, *Solution techniques for some allocation problems, Mathematical* Programming, 25 (1983), pp. 13-24.

[Geo70] A. M. Geoffrion, *Elements of large-scale mathematical programming. part II: Synthesis of algorithms and bibliography,* Management Science, 16 (1970), pp. 652-691.

[HKL80] R. V. Helgason, J. L. Kennington, and H. Lall,*A polynomially bounded algorithm for a singly constrained quadratic program, Mathematical Programming,* 18 (1980), pp. 338-343.

[IbK88] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches,* no. 4 in Foundations of Computing Series, The MIT Press, Cambridge, MA, 1988.

[KIM79] N. Katoh, T. Ibaraki, and H. Mine, *A polynomial time algorithm for the resource allocation problem with a convex objective function,* Journal of the Operational Research Society, 30 (1979), pp. 449-455.

[Kiw07] K.C Kiwiel, *On Linear-Time Algorithms for the Continuous Quadratic knapsack problem*, Mathmatical programming, Ser. A, Vol. 134, No 3. pp. 149-154, 2007

[Kiw08] K.C Kiwiel, *Breakpoint searching algorithms for continuous quadratic knapsack problem*, , Vol. 112, No 2 pp. 473-491, 2008

[Kiw08b] K.C Kiwiel, *Variable Fixing Algorithms for Continuous Quadratic Knapsack Problem*, J Optim Theory Appl 136, pp. 445-458, 2008

[Knu98] D. E. Knuth, *Sorting and Searching, 2nd edn. The Art of Computer Programming,* vol. III. Addison-Wesley, Reading (1998)

[KoL98] M. S. Kodialam and H. Luss, *Algorithms for separable nonlinear resource allocation problems,* Operations Research, 46 (1998), pp. 272-284.

[Koo99] B. O. Koopman, *Search and Screening. General Principles With Historical Applications*, Military Operations Research Society, Alexandria, VA, revised ed., 1999.

[LMOM07] T. Larsson, J. Marklund, C. Olssone, M. Patriksson, *Convergent Lagrangian heuristics for nonlinear minimum cost network flows,* Operations Research, 189 (2008) pp. 324-346.

[LuG75] H. Luss and S.K. Gupta, *Allocation of effort resources among Competing Activities*, Operation Research, 23 (1975) pp. 360-366

[MaK93] B. M. Maloney and C. M. Klein *Constrained multi-item inventory systems: An implicit approach,* Computations and Operational Research, 20 (1993), pp. 639-649.

[Mar52] H. M. Markowitz, *Portfolio selection,* Journal of Finance 7 (1952), pp. 77-91.

[MaT90] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations,* New York: Wiley, 1990.

[MGB81] J. J. Moré, B. S. Garbow and K. E. Hillstrom *Testing Unconstrained Optimization Software,* ACM Transactions on Math. Softwares, vol 7, No.1 (1981), pp. 17-41.

[NiZ92] S.S Nielsen and S. A. Zenios, *Massively parallel algorithms for singly constrained convex programs*, ORSA Journal on Computing, 4 (1992), pp. 166-181.

[Pat08] M. Patriksson, *A survey on the continuous nonlinear resource allocation problem*, European Journal of Operational Research, Volume 185, p.1-46, 2008 Press, Princeton, NJ, 1970.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery *Numerical Recipes in Fortran,* 2nd ed. Cambridge University Press (1992).

[RJL92] A. G. Robinson,N. Jiang and C. S. Lemke , *On the continuous quadratic knapsack problem,* Mathematical programming 55 (1992) pp. 99-108.

[San71] L. Sanathanan, *On an allocation problem with multistage constraints,* Operations Research, 18 (1971), pp. 1747-1663.

[Sha63] W. F. Sharpe, *A simplified model for portfolio analysis, Management Science,* 9 (1963), pp. 277-293.'

[Sri63] K. S. Srikantan, *A problem in optimum allocation,* Operations Research, 18 (1963), pp. 265-273.

[Ste01] S. M. Stefanov, *Convex separable minimization subject to bounded variables,* Computational Optimization and Applications, 18 (2001), pp. 27-48.

[Sto81] L.D. Stone Review (Untitled) , (Review of Searching an Screening: General Principles with Historical Applications) SIAM Review vol. 23 No. 4 (1981)

[ZeC91] S. A. Zenios and Y. Censor, *Massively parallel row-action algorithms for some nonlinear transportation problems,* SIAM Journal on Optimization, 1 (1991), pp. 373-400.

[Zip80] P. H. Zipkin, *Simple ranking methods for allocation of one resource,* Managemant Science, 26 (1980), pp. 34-43.

A CPU-times for the numerical experiments

We post the results for the computatinal experiment for DBR, MMB and the quasi-Newton method. The exponentials denotes the number of problem instances that where unsolved by the algorithm.

A.1 The theory of search problem

Table 2: CPU-times for MMB for solving problem (67). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0184	0.0396	0.0798	0.2098	0.4163	0.8528
$10-20\%$	0.0205	0.0432	0.0870	0.2195	0.4536	0.9157
$20-30\%$	0.0231	0.0477	0.0985	0.2522	0.5017	1.0350
$30-40\%$	0.0252	0.0532	0.1090	0.2804	0.5635	1.1361
$40-50\%$	0.0285	0.0583	0.1194	0.3108	0.6377	1.3069
$50-60\%$	0.0321	0.0656	0.1384	0.3512	0.7188	1.4672
$60-70\%$	0.0351	0.0723	0.1504	0.3930	0.8011	1.6047
$70-80\%$	0.0382	0.0798	0.1659	0.4254	0.8739	1.7491
$80-90\%$	0.0416	0.0859	0.1804	0.4652	0.9487	1.9048
$90-100\%$	0.0453	0.0934	0.1956	0.5036	1.0218	2.0885
Average	0.0308	0.0639	0.1324	0.3411	0.6937	1.4061

Table 3: CPU-times for DBR for solving problem (67). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0222	0.0467	0.0939	0.2389	0.4651	0.9546
$10-20\%$	0.0226	0.0481	0.0967	0.2403	0.4800	0.9649
$20-30\%$	0.0235	0.0486	0.0985	0.2450	0.5032	1.0075
$30-40\%$	0.0239	0.0499	0.1017	0.2544	0.5230	1.0526
$40-50\%$	0.0246	0.0516	0.1055	0.2605	0.5419	1.1001
$50-60\%$	0.0252	0.0533	0.1099	0.2756	0.5683	1.1263
$60-70\%$	0.0259	0.0543	0.1116	0.2800	0.5799	1.1681
$70-80\%$	0.0262	0.0537	0.1116	0.2860	0.5855	1.1714
$80-90\%$	0.0258	0.0534	0.1143	0.2853	0.5806	1.1651
$90-100\%$	0.0253	0.0528	0.1079	0.2798	0.5557	1.1118
Average	0.0245	0.0512	0.1051	0.2645	0.5383	1.0823

Table 4: CPU-times for ZN for solving problem (67). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.7418	4.1150	3.1260	9.3488	21.4246	42.2321
$10-20\%$	0.2524	0.4186	1.2671	2.9730	4.5789	12.4133
$20-30\%$	0.1955	0.2749	0.5175	1.7088	3.4769	7.4375
$30-40\%$	0.1491	0.2781	0.5414	1.5100	2.7338	5.4466
$40-50\%$	0.1087	0.2181	0.4349	1.4146	2.5865	6.7664
$50-60\%$	0.0948	0.2057	0.4212	1.1176	2.4756	5.0269
$60-70\%$	0.0760	0.1591	0.3184	0.8111	1.8395	3.8852
$70-80\%$	0.0754	0.1418	0.2986	0.8252	1.5393	3.4454
$80-90\%$	0.0612	0.1240	0.2307	0.6251	1.4669	2.6413
$90-100\%$	0.0432	0.0887	0.1800	0.4901	0.9935	1.9442
Average	0.1798	0.6024	0.7336	2.0824	4.3116	9.1239

A.2 The quadratic problem

Table 5: CPU-times for MMB for solving the quadratic problem (63). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0129	0.0273	0.0564	0.1535	0.3106	0.6549
$10-20\%$	0.0146	0.0309	0.0677	0.1746	0.3834	0.7860
$20-30\%$	0.0168	0.0355	0.0734	0.2017	0.4306	0.8471
$30-40\%$	0.0188	0.0391	0.0840	0.2208	0.4627	0.9055
$40-50\%$	0.0205	0.0433	0.0942	0.2448	0.5204	1.0727
$50-60\%$	0.0232	0.0476	0.1037	0.2697	0.5768	1.1659
$60-70\%$	0.0248	0.0518	0.1125	0.2958	0.6299	1.2380
$70-80\%$	0.0264	0.0564	0.1228	0.3293	0.6865	1.3808
$80-90\%$	0.0295	0.0612	0.1315	0.3526	0.7287	1.4749
$90-100\%$	0.0327	0.0691	0.1505	0.3981	0.8096	1.6547
Average	0.0220	0.0462	0.0997	0.2641	0.5539	1.1181

Table 6: CPU-times for DBR for solving the problem (63). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0099	0.0199	0.0446	0.1163	0.2457	0.5025
$10-20\%$	0.0100	0.0207	0.0439	0.1184	0.2366	0.4926
$20-30\%$	0.0100	0.0203	0.0435	0.1144	0.2449	0.4840
$30-40\%$	0.0103	0.0207	0.0448	0.1136	0.2403	0.4840
$40-50\%$	0.0100	0.0217	0.0450	0.1178	0.2403	0.4632
$50-60\%$	0.0106	0.0212	0.0430	0.1143	0.2393	0.4949
$60-70\%$	0.0101	0.0219	0.0452	0.1083	0.2373	0.4599
$70-80\%$	0.0101	0.0214	0.0454	0.1167	0.2435	0.4883
$80-90\%$	0.0102	0.0201	0.0459	0.1121	0.2218	0.4737
$90-100\%$	0.0098	0.0202	0.0424	0.1026	0.2122	0.4088
Average	0.0101	0.0208	0.0444	0.1134	0.2362	0.4752

Table 7: CPU-times for ZN for solving the problem (63). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0796	0.3027	1.2227	2.8052	5.2094	10.0891
$10-20\%$	0.0515	0.0892	0.2025	0.5031	1.3150	2.3668
$20-30\%$	0.0337	0.0726	0.1416	0.5276	0.8453	1.9715
$30-40\%$	0.0324	0.0609	0.1434	0.4307	0.7762	1.6162
$40-50\%$	0.0290	0.0623	0.1353	0.3424	0.7638	1.3782
$50-60\%$	0.0265	0.0562	0.1152	0.3303	0.6331	1.2935
$60-70\%$	0.0229	0.0514	0.1123	0.2863	0.5983	1.1649
$70-80\%$	0.0228	0.0476	0.1003	0.2792	0.5523	1.0738
$80-90\%$	0.0192	0.0419	0.0893	0.2333	0.4558	0.9815
$90-100\%$	0.0186	0.0393	0.0810	0.2052	0.4331	0.8531
Average	0.0336	0.0824	0.2343	0.5943	1.1582	2.2789

A.3 The stratified sampling problem

Table 8: CPU-times for MMB for solving problem (65). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0179	0.0417	0.0791	0.2108	0.4372	0.9604
$10-20\%$	0.0200	0.0436	0.0902	0.2289	0.4731	1.0263
$20-30\%$	0.0221	0.0467	0.0980	0.2581	0.5285	1.1010
$30-40\%$	0.0227	0.0493	0.1051	0.2754	0.5697	1.1414
$40-50\%$	0.0250	0.0537	0.1131	0.2941	0.6133	1.2642
$50-60\%$	0.0270	0.0560	0.1222	0.3135	0.6672	1.3309
$60-70\%$	0.0285	0.0612	0.1324	0.3451	0.7041	1.4412
$70-80\%$	0.0318	0.0658	0.1416	0.3745	0.7620	1.5477
$80-90\%$	0.0330	0.0701	0.1543	0.3975	0.8275	1.6631
$90-100\%$	0.0362	0.0765	0.1676	0.4324	0.8857	1.7934
Average	0.0264	0.0565	0.1203	0.3130	0.6468	1.3270

Table 9: CPU-times for DBR for solving problem (65). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0105	0.0220	0.0459	0.1195	0.2433	0.4854
$10-20\%$	0.0109	0.0231	0.0480	0.1227	0.2473	0.5095
$20-30\%$	0.0110	0.0236	0.0495	0.1267	0.2590	0.5210
$30-40\%$	0.0116	0.0240	0.0496	0.1291	0.2649	0.5365
$40-50\%$	0.0117	0.0246	0.0522	0.1305	0.2666	0.5447
$50-60\%$	0.0121	0.0246	0.0518	0.1350	0.2699	0.5505
$60-70\%$	0.0127	0.0270	0.0542	0.1361	0.2836	0.5672
$70-80\%$	0.0132	0.0265	0.0550	0.1372	0.2837	0.5596
$80-90\%$	0.0130	0.0267	0.0551	0.1385	0.2851	0.5479
$90-100\%$	0.0122	0.0246	0.0510	0.1283	0.2610	0.5457
Average	0.0119	0.0247	0.0512	0.1303	0.2664	0.5368

Table 10: CPU-times for ZN for solving problem (65). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.2090^{2}	0.6756^{2}	0.8313^{1}	2.3624^{5}	3.6534^{6}	22.9621^{2}
$10-20\%$	0.1523^{3}	0.2088^{5}	0.4253^{5}	1.0248^{3}	6.4086^{5}	10.7805^{2}
$20-30\%$	0.0952^{2}	0.4674^{3}	1.0834^{3}	1.9378^{2}	2.6699^{4}	13.1352^{4}
$30-40\%$	0.0770^{6}	0.3376^{4}	0.3182^{6}	1.7652^{3}	3.4411^{5}	3.5835^{5}
$40-50\%$	0.1975^{6}	0.3471^{5}	0.9239^{6}	0.8389^{5}	3.8029^{3}	6.4066^{3}
$50-60\%$	0.1715^{1}	0.2070^{4}	0.6809^{5}	0.7411^{1}	3.0716^{1}	9.6600^{6}
$60-70\%$	0.0999^{2}	0.2489	0.4742^{1}	1.6995^{2}	1.8511^{2}	8.8821^{2}
$70-80\%$	0.1024	0.3224	0.4062	1.9369	2.1858	4.0281
$80-90\%$	0.0868	0.2742	0.6412	1.1045	1.6544	6.4222
$90-100\%$	0.0186	0.0367	0.3077	1.9830	1.6323	3.3603
Average	0.1210	0.3126	0.6092	1.5394	3.0371	8.9221

A.4 The negative entropy problem

Table 11: CPU-times for MMB for solving problem (68). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0171	0.0364	0.0751	0.1983	0.4022	0.8353
$10-20\%$	0.0183	0.0400	0.0823	0.2108	0.4354	0.9031
$20-30\%$	0.0199	0.0416	0.0873	0.2308	0.4698	0.9661
$30-40\%$	0.0214	0.0448	0.0921	0.2457	0.5192	1.0337
$40-50\%$	0.0230	0.0470	0.0971	0.2648	0.5338	1.0962
$50-60\%$	0.0240	0.0512	0.1049	0.2794	0.5752	1.1640
$60-70\%$	0.0256	0.0534	0.1104	0.2955	0.6088	1.2398
$70-80\%$	0.0263	0.0559	0.1160	0.3103	0.6405	1.3191
$80-90\%$	0.0277	0.0584	0.1218	0.3266	0.6831	1.3833
$90-100\%$	0.0298	0.0612	0.1297	0.3468	0.7243	1.4910
Average	0.0233	0.0490	0.1017	0.2709	0.5592	1.1432

Table 12: CPU-times for DBR for solving problem (68). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0128	0.0265	0.0545	0.1454	0.2948	0.6159
$10-20\%$	0.0135	0.0281	0.0579	0.1510	0.3111	0.6308
$20-30\%$	0.0141	0.0288	0.0593	0.1563	0.3222	0.6604
$30-40\%$	0.0144	0.0298	0.0611	0.1596	0.3354	0.6831
$40-50\%$	0.0150	0.0304	0.0637	0.1679	0.3444	0.6728
$50-60\%$	0.0151	0.0308	0.0635	0.1691	0.3387	0.6866
$60-70\%$	0.0154	0.0310	0.0635	0.1703	0.3413	0.6934
$70-80\%$	0.0146	0.0309	0.0649	0.1641	0.3421	0.7115
$80-90\%$	0.0154	0.0302	0.0612	0.1632	0.3334	0.6906
$90-100\%$	0.0138	0.0307	0.0605	0.1562	0.3162	0.6486
Average	0.0144	0.0297	0.0610	0.1603	0.3280	0.6694

Table 13: CPU-times for ZN for solving problem (68). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.2851	0.3737	1.4003	2.3651	6.8572	11.5980
$10-20\%$	0.0758	0.1422	0.2610	1.3356	1.5343	4.1492
$20-30\%$	0.0541	0.1104	0.1951	0.6915	1.5987	2.4485
$30-40\%$	0.0387	0.0958	0.1938	0.5216	0.9344	2.3886
$40-50\%$	0.0331	0.0706	0.1859	0.4051	1.0056	1.4386
$50-60\%$	0.0301	0.0514	0.1427	0.3526	0.7266	1.5795
$60-70\%$	0.0259	0.0559	0.1209	0.2851	0.6340	1.3041
$70-80\%$	0.0266	0.0512	0.1030	0.2838	0.6065	1.1961
$80-90\%$	0.0218	0.0458	0.0952	0.2682	0.5302	1.1000
$90-100\%$	0.0201	0.0418	0.0865	0.2202	0.4389	0.8910
Average	0.0611	0.1039	0.2784	0.6729	1.4866	2.8094

A.5 The sampling problem

Table 14: CPU-times for MMB for solving problem (66). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0138	0.0288	0.0627	0.1729	0.3557	0.7382
$10-20\%$	0.0146	0.0318	0.0684	0.1848	0.3849	0.7910
$20-30\%$	0.0162	0.0334	0.0721	0.1948	0.4116	0.8382
$30-40\%$	0.0175	0.0369	0.0778	0.2048	0.4416	0.8853
$40-50\%$	0.0189	0.0396	0.0815	0.2217	0.4658	0.9470
$50-60\%$	0.0199	0.0416	0.0879	0.2345	0.4903	1.0052
$60-70\%$	0.0208	0.0443	0.0928	0.2510	0.5165	1.0584
$70-80\%$	0.0220	0.0467	0.0981	0.2600	0.5408	1.1074
$80-90\%$	0.0233	0.0491	0.1031	0.2780	0.5788	1.1707
$90-100\%$	0.0245	0.0522	0.1121	0.3017	0.6281	1.2494
Average	0.0191	0.0404	0.0856	0.2304	0.4814	0.9791

Table 15: CPU-times for DBR for solving problem (66). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.0098	0.0199	0.0454	0.1263	0.2845	0.5844
$10-20\%$	0.0101	0.0215	0.0471	0.1358	0.2846	0.5959
$20-30\%$	0.0109	0.0215	0.0473	0.1302	0.2827	0.5729
$30-40\%$	0.0112	0.0230	0.0493	0.1296	0.2882	0.5762
$40-50\%$	0.0113	0.0226	0.0474	0.1304	0.2760	0.5708
$50-60\%$	0.0111	0.0231	0.0477	0.1320	0.2751	0.5626
$60-70\%$	0.0115	0.0224	0.0472	0.1254	0.2665	0.5553
$70-80\%$	0.0114	0.0229	0.0471	0.1283	0.2747	0.5515
$80-90\%$	0.0113	0.0235	0.0472	0.1253	0.2587	0.5213
$90-100\%$	0.0109	0.0239	0.0467	0.1231	0.2438	0.4947
Average	0.0109	0.0224	0.0472	0.1286	0.2734	0.5586

Table 16: CPU-times for ZN for solving problem (66). Each value is the mean of 10 randomized computations.

$card(\mathbf{H})/n$	50000	100000	200000	500000	1000000	2000000
$0-10\%$	0.2700^{3}	0.6880^{1}	1.2740^{1}	4.2300^{2}	5.5869^{2}	18.5861^{1}
$10-20\%$	0.2325^{1}	0.5413^{1}	1.1520^{1}	2.3728	6.0567^{2}	15.3734^{2}
$20-30\%$	0.3014	1.2775	0.9126^{2}	2.6608	8.3284^{2}	8.6787
$30-40\%$	0.5222	0.7717	2.1255	3.9237	14.7029^{1}	18.0526
$40-50\%$	0.2517	0.9015	1.4935	5.1415	7.6900	13.4152
$50-60\%$	0.3143	0.7356	1.5437	5.8396	8.6370	15.0744
$60-70\%$	0.2532	0.9033	1.4728	2.4379	8.0379	12.4096
$70-80\%$	0.2584	0.5863	1.7102	2.5334	8.0650	12.9497
$80-90\%$	0.1048	0.2808	0.5144	1.0797	3.3580	5.5380
$90-100\%$	0.0200	0.0409	0.0940	0.2349	0.8587	0.9559
Average	0.2528	0.6727	1.2293	3.0454	7.1321	12.1033