Pontifícia Universidade Católica
do Rio de Janeiro

João Pedro Teixeira Brandão

Parametric search for variants of nested resource allocation problems

Dissertação de Mestrado

Thesis presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Thibaut Victor Gaston Vidal

Rio de Janeiro
February 2019

PUC-Rio - Certificação Digital Nº 1621891/CA

All rights reserved.

João Pedro Teixeira Brandão

Bachelor's in Applied Mathematics (2016) at Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Worked at Tecgraf from 2011 to 2012 in the Engineering Automation Group and took part in the first Apple Developer Academy program offered by PUC-Rio in 2015.

PUC-Rio - Certificação Digital Nº 1621891/CA

Bibliographic data

Brandão, João Pedro Teixeira

Parametric search for variants of nested resource allocation problems / João Pedro Teixeira Brandão; advisor: Thibaut Victor Gaston Vidal. – 2019.

52 f.: il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2019

Inclui bibliografia

1. Informática – Teses. 2. Busca paramétrica. 3. Restrições aninhados. 4. Alocação de recurso. 5. Otimização. 6. Programação linear. I. Vidal, Thibaut Victor Gaston. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

To my advisor, Professor Thibaut Vidal for his dedication and generosity for sharing his knowledge with me.

To my parents and brother, Sonia and (Luiz)^2 Brandão for their unconditional support.

To PUC's Informatics Department for their support in providing an environment that fosters knowledge sharing.

To my colleagues at GoBlock for being supportive of my endeavors.

To Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for financing this research under grant 134797/2016-7.

PUC-Rio - Certificação Digital Nº 1621891/CA

Abstract

Brandão, João Pedro Teixeira; Vidal, Thibaut Victor Gaston (Advisor). **Parametric search for variants of nested resource allocation problems**. Rio de Janeiro, 2019. 52p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The Resource Allocation Problems seek to find an optimal repartition of resources into a fixed number of areas. In this thesis, we consider a resource allocation problem with a linear objective and two distinct sets of constraints: a set of nested constraints, where the partial sums of the decision variables are limited from above and a linear constraint that defines a hyperplane. We propose a weakly and a strongly polynomial algorithm. The weakly polynomial algorithm requires certain assumptions of the data and runs in $O(n \log n \log \frac{|\Lambda|}{|I|})$ time, where n is the number of decision variables, Λ is an interval in the dual space, and $|I|$ relates to the precision of the data. The strongly polynomial algorithm is based on Megiddo's parametric search technique, and obtains a complexity of $O(n \log n)$. These are large improvements upon the $O(n^3/\log n)$ complexity of the generic Interior Point Method. In addition, an experimental analysis was carried out and the algorithms showed to be more efficient and produced optimal solutions for problem instances with up to 1,000,000 variables.

PUC-Rio - Certificação Digital Nº 1621891/CA

Keywords

Parametric Search; Nested Constraints; Resource Allocation; Optimization; Linear Programing.

Resumo

Brandão, João Pedro Teixeira; Vidal, Thibaut Victor Gaston. **Busca paramétrica para variantes do problema de alocação de recurso aninhado**. Rio de Janeiro, 2019. 52p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os problemas de alocação de recurso procuram encontrar uma repartição ideal de recursos a um número fixo de áreas. Nesta dissertação, consideramos um problema de alocação de recurso com uma função objetiva linear e dois conjuntos distintos de restrições: um conjunto de restrições aninhados, onde as somas parciais das variáveis de decisão são limitadas por cima e uma restrição linear que define um hiperplano. Propomos um algoritmo fracamente e um fortemente polinomial. O algoritmo fracamente polinomial requer algumas suposições sobre os dados e possui complexidade de $O(n \log n \log \frac{|\Lambda|}{|I|})$, onde n é o número de variáveis, Λ é um intervalo no espaço dual, e $|I|$ está relacionado com a precisão dos dados. O algoritmo fortemente polinomial é baseado na técnica de busca paramétrica de Megiddo e obtém uma complexidade $O(n \log n)$. As complexidades obtidas são superiores à complexidade do método genérico de Pontos Interiores, $O(n^3/\log n)$. Além disso, uma análise experimental foi realizada e os algoritmos mostraram-se mais eficientes e produziram soluções ótimas para instâncias de problemas com até 1.000.000 variáveis.

PUC-Rio - Certificação Digital Nº 1621891/CA

Palavras-chave

Busca paramétrica; Restrições aninhados; Alocação de recurso; Otimização; Programação linear.

Table of contents

PUC-Rio - Certificação Digital Nº 1621891/CA

List of figures

PUC-Rio - Certificação Digital Nº 1621891/CA

List of tables

PUC-Rio - Certificação Digital Nº 1621891/CA

1
Introduction

Resource Allocation Problems, seek to find an optimal repartition of resources into a fixed number of areas. The resource can represent, for example, time, money, or workforce. The areas can represent assets, locations, or tasks. An objective value function determines the cost of choosing to allocate a quantity of the resource to a particular area. The optimization goal is to find the allocation of resource that has the smallest value of the objective function out of all other possible allocations. The simple RAP, aims to minimize a separable objective function subject to one linear resource constraint. Most of the research on the RAP focuses on extensions of the problem with one additional constraint, due to the ample applications of the model. Several of these variants are reviewed in (Patriksson2008), (Patriksson2015), and (Hochbaum1994). However, variants with two or more distinct sets of constraints have not been the subject of dedicated studies up to now. The simple RAP can be formally defined as:

$$\min \ \mathbf{F}(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i) \tag{1-1}$$

$$\sum_{i=1}^{n} x_i = B \tag{1-2}$$

$$x_i \geqslant 0, \qquad i \in \{1, ..., n\} \tag{1-3}$$

The objective function, defined in (1-1), is separable, i.e. the cost of allocating resource to a variable is determined solely by the quantity allocated to that variable, x_i. The resource constraint is defined by Equation (1-2), the nonnegativity constraint is defined in Inequality (1-3).

In this study, however, we focus on the model shown by Equations (1-

PUC-Rio - Certificação Digital Nº 1621891/CA

4)–(1-8):

$$F(\mathbf{x}) = \min \sum_{i=1}^{n} c_i x_i \tag{1-4}$$

$$\text{s.t.} \sum_{i \in J_k} x_i \leqslant b_k, \qquad k \in \{1, ..., m\} \tag{1-5}$$

$$\sum_{i=1}^{n} x_i = B_1 \tag{1-6}$$

$$\sum_{i=1}^{n} p_i x_i = B_2 \tag{1-7}$$

$$x_i \geqslant 0, \qquad i \in \{1, ..., n\} \tag{1-8}$$

The objective function (1-4) is linear. The set of inequalities, defined by (1-5), are the nested constraints. The sets J_k are in a nested format where $J_{k-1} \subset J_k$, and $J_m = \{1, ..., n\}$, and $b_m = B_1$. The resource constraint is defined in Equation (1-6). In Equation (1-7), the linear constraint is defined. Lastly, the nonnegativity constraint is defined by Inequality (1-8). We will denote the problem defined by Equations (1-4)–(1-8) as RAP-LB-L, and the RAP-LB-L without the extra linear constraint (1-7) as the RAP-LB.

The RAP-LB-L is connected to inventory management subproblem of the Inventory Routing Problem (IRP), see (Hartl2016) and (Coelho2014), and the Bike Sharing Problem, see (Chemla2013). By studying and understanding the formulation (1-4)–(1-8), we hope to make the first steps towards efficiently solving the more complex settings in the future.

We propose two algorithms based on Lagrangean Relaxation, a weakly and a strongly polynomial algorithm. Under certain assumptions of the data we develop a weakly polynomial algorithm with complexity $O(n \log n \log \frac{|\Lambda|}{|I|})$, where n is the number of variables, Λ is the search interval in the dual space, and $|I|$ is a precision parameter. The proposed strongly polynomial algorithm has no assumptions of the data and has complexity $O(n \log n)$.

The strongly polynomial algorithm is a variant of (Cole1989)'s solution to the slope selection problem. The solution applies the parametric search technique of (Megiddo1983) to develop an optimal algorithm. Both algorithms achieve better complexities than the Interior Point method (IP), introduced by (Karmarkar1984) and improved to $O(n^3/\log n)$ by (Anstreicher1999), which to the best of our knowledge is the latest improvement.

The structure of this dissertation is as follows. In Chapter 2, we review the literature on RAP and discuss the different approaches as well as the results attained. We also review the literature on the parametric search technique and slope selection problem. Chapter 3 discusses the optimality conditions

PUC-Rio - Certificação Digital Nº 1621891/CA

of the RAP-LB-L and characterize the dual optimal, as well as detailing a method for extracting the primal optimal from the dual optimal. Chapters 4 and 5 present the algorithms and a proof of their correctness and complexity. Chapter 6 reports our experiments and results carried out with the weakly and strongly polynomial algorithm. Finally Chapter 7 concludes.

PUC-Rio - Certificação Digital Nº 1621891/CA

2
Literature Review

2.1
Resource Allocation Problem

Though there has been extensive research on variants of the RAP concerning only one set of constraints as surveyed by (Patriksson2008) and (Patriksson2015), variants with two sets of constraints have been less explored. The RAP variant with quadratic objective with only constraint (1-6) and a box constraint, where each variable is limited to a finite range, has been shown to be solvable in linear time by (Brucker1984). A different approach based on the prune and search technique developed by (Megiddo1984), allowed (Megiddo1993) to show that the same problem but with a constant number of linear constraints can also be solved in linear time. The paper shows that more general class of separable quadratic programming problems can be solved in linear time, where the problem worked on by (Brucker1984) is only a special case.

Almost all RAP studied in the literature have unimodular constraints, as seen in (Ibaraki1988). (Hochbaum1994) show that several subclasses of unimodular constraint problems cannot be solved in a strongly polynomial time in the presence of continuous variables. By using a result from (Renegar1987), (Hochbaum1994) demonstrated a lower bound complexity of $\Omega(\sqrt{\log\log(\frac{B}{\epsilon})})$ for a special case of the general RAP when the floor operation is allowed. The ϵ term is the desired accuracy of the solution. The solution for the continuous case is then an ϵ-approximate solution, where $\mathbf{x}^\epsilon$ is ϵ-approximate if there exists an optimal solution $\mathbf{x}^*$ such that $||\mathbf{x}^\epsilon - \mathbf{x}^*||_\infty \leqslant \epsilon$. Another significant result from (Hochbaum1994) is the proximity theorem which is applied to the greedy algorithm. At each iteration the greedy algorithm chooses to increment a variable by one, such that the increment results in the greatest increase of the objective function. The proximity theorem allows the increment to be arbitrary. This results in an $O(n(\log n + F)\log(\frac{B}{n}))$ algorithm for the general integer problem and $O(n(\log n + F)\log(\frac{B}{n\epsilon}))$ for the continuous case, where F is the number of operations required to check the feasibility of an increment in the solution, and B is the resource.

PUC-Rio - Certificação Digital Nº 1621891/CA

Some variants of the RAP are discussed in (Hochbaum1994). The tree RAP variant has m additional inequalities. The inequalities are on subsets of the set $J_m = \{1, ..., n\}$. The subsets J_k form a tree structure where the root is the set J_m, and nodes connected by edges are proper subsets. One special case of the tree RAP variant is the generalized upper bound resource allocation problem (GUB), where the subsets of J_m form a partition. Another tree variant is the nested RAP, where the tree structure is a path. It would be a special case of the RAP-LB-L without constraint (1-7).

An equivalent variant with the same nested structure is the RAP-LB, which is also known as the RAP with linear ascending constraints. The RAP-LB can be reduced to RAP-UB in linear time. In this problem, the objective function is convex and there are m inequalities on top of subsets of J_m that have a nested structure. The inequalities are bounded from above. The study of this class of problems is motivated by its applications in signal processing and communications, see (D'Amico2014) and (Akhil2014).

A more general nested structure, named RAP-NC, where the sums of variables in J_k are bounded from below and above is addressed in (Vidal2018). The proposed decomposition algorithm is built on previous work from, (Vidal2016). The achieved complexity is $O(n \log m \log \frac{nB}{\epsilon})$ for the continuous case and $O(n \log m \log B)$ for the integer case. In particular (Vidal2018) obtains an $O(n \log m)$ complexity for cases when the objective function is either linear or quadratic.

Finally, a subproblem of both the IRP and the bike sharing can be modelled by a RAP with two different sets of nested constraint. This variant has been studied in (Hartl2016) and (Chemla2013). In (Chemla2013), the authors study a bike sharing system and deals with a generalization of several pickup and delivery problems. In the bike sharing system, there are stations scattered accross the city. Each station has bikes where people can pick up a bike and drop it off at another station. The subproblem is then, given a fixed route, find the best way to rebalance the stations.

2.2 Parametric Search and The Slope Selection Problem

Parametric search is a technique that uses parallel algorithms to design efficient serial algorithms. To apply the technique to problem $\mathcal{B}$, this problem must depend monotonically on a parameter, λ. Due to this monotonicity, an algorithm that employs binary search can be made to solve $\mathcal{B}$, however it would not be strongly polynomial. Parametric search, on the other hand, constructs strongly polynomial algorithms.

PUC-Rio - Certificação Digital Nº 1621891/CA

Parametric search was first formally introduced by (Megiddo1983). However, Megiddo had already applied the technique in his earlier work. In (Megiddo1981) the technique is applied on a RAP, in which the objective function is the ratio between the sum of separable concave functions and a sum of separable convex functions. The only constraint, other than the resource constraint, is that the variables are positive integers.

Further improvements upon the basic technique were achieved by (Cole1987). The improvement enables a complexity reduction of a factor of $O(\log n)$ under special cases of the parametric search framework. However, the gain in theoretical efficiency comes at the cost of further complicating the parametric search implementation.

Megiddo's parametric search has been applied extensively to solve geometric optimization problems. Some of the applications can be found in (Chazelle1992), (Agarwal1994) and (Agarwal1998). The parametric search has also been used to find the Theil-Sen estimator in (Sen1968). The Theil-Sen estimator is used to robustly fit a line to a sample of points by choosing the median slope of the points. It is less sensitive to outliers than the traditional linear regression. The problem of finding the median slope among a set of n points is a particular case of a more general problem known as the Slope Selection Problem (SSP).

In the SSP, there are n points in $\Re^2$ and for every pair of points, a line segment is drawn. The task is to find the pair of points that define a line segment that has the k-th shallowest slope out of all the other line segments. The problem was first solved by (Cole1989) using parametric search. In the paper, the authors proved a lower bound for the complexity of $O(n \log n)$. The authors first achieved a $O(n \log^3 n)$ complexity. By using the optimization from (Cole1987), they reduced the complexity to $O(n \log^2 n)$. Finally, by developing a linear time procedure to substitute another $O(n \log n)$ time procedure, the overall complexity was reduced to the optimal $O(n \log n)$. There are other algorithms that solve the problem but do not use parametric search, see (Katz1993) and (Brönnimann1998). Instead of using parametric search, the former uses expander graph (Lubotzky2012), while the latter uses ϵ-nets. Both algorithms, however, use the linear time procedure developed in (Cole1987).

Practical implementations of Megiddo's technique took longer than their theoretical definitions to become a reality. (vanOostrum2004) implemented a sorting based application of parametric search. The quicksort algorithm was used. Although it has a worse case complexity of $O(n^2)$, its simplicity of implementation and average complexity of $O(n \log n)$ lead to efficient implementations. Yet, due to quicksort's algorithm, Cole's optimization cannot be im-

PUC-Rio - Certificação Digital Nº 1621891/CA

plemented under this framework, as noted in (vanOostrum2004). The authors also argue that under certain realistic assumptions, the optimization may be unnecessary. Finally, (Goodrich2013), proposed a practical implementation of Cole's technique that has the same complexity with high probability. They substitute the AKS sorting network that was used in (Cole1987) with the box sort (Reischuk1985).

PUC-Rio - Certificação Digital Nº 1621891/CA

3
Preliminaries

This chapter states some basic results about the RAP with nested constraints, the optimality conditions of our problem, a procedure that obtains a primal optimal solution from the dual optimal, and detail a procedure to check for problem feasibility. As such, we will assume feasibility throughout. The results in this chapter will serve to prove correctness of our weakly and strongly polynomial algorithms.

Bold face notation will be used to denote vectors, i.e. $\mathbf{x} = (x_1, x_2, ..., x_n)$, while normal font will be used for scalar variables. We assume, without loss of generality, that all coefficients and constants are integers.

PUC-Rio - Certificação Digital Nº 1621891/CA

3.1
Known Results on the RAP-LB and RAP-NC

Problem (1-4) with constraints (1-5), (1-6) and (1-8) is the RAP-LB. It is known that this problem can be trivially solved, in $O(n)$ time, by the greedy algorithm defined in Algorithm 3.1.1.

Algorithm 3.1.1: Greedy(λ)

```
1 x ← 0;
2 for k ← m to 1 do
3 |   Δ ← b_k − b_{k−1};
4 |   i_min ← arg min_i {c_i | i ∈ (J_k − J_{k−1}) ∪ {i_min}};
5 |   x_{i_min} ← x_{i_min} + Δ
6 end
```

A special case of the problem addressed in (Vidal2018) is defined in Equations (3-1)–(3-4). The problem is a generalization of the RAP-LB.

$$\min \sum_{i=1}^{n} c_i x_i \tag{3-1}$$

$$\text{s.t. } a_i \leqslant \sum_{i \in J_k} x_i \leqslant b_k, \qquad k \in \{1, ..., m\} \tag{3-2}$$

$$\sum_{i=1}^{n} x_i = B_1 \tag{3-3}$$

$$c_i \leqslant x_i \leqslant d_i \tag{3-4}$$

$$J_1 \subset J_2 \subset ... \subset J_m, \qquad J_m = \{1, ..., n\} \tag{3-5}$$

The proposed divide-and-conquer algorithm runs in $O(n \log m)$ time and is based on monotonicity arguments. In the algorithm, the solutions found at lower levels of recursion are shown to be tighter bounds than the nested constraints. This allows to simplify the problem and remove the nested contraints (3-2), reducing the formulation to simple RAP, i.e. the problem defined by Equations (1-1)–(1-3). With this, efficient RAP algorithms can be used at each level of the recursion, such as the procedure developed by (Brucker1984).

PUC-Rio - Certificação Digital Nº 1621891/CA

3.2 Lagrangean Relaxation

The Lagrangean Relaxation (LR) of problem RAP-LB-L defined in (1-4)–(1-8), denoted as $L(\mathbf{x}, \lambda)$, is defined in Equations (3-6)–(3-9):

$$L(\mathbf{x}, \lambda) = \sum_{i=1}^{n} c_i x_i + \lambda\Big(\sum_{i=1}^{n} p_i x_i - B_2\Big) \tag{3-6}$$

$$\text{s.t. } \sum_{i \in J_k} x_i \leqslant b_k, \qquad k \in \{1, ..., m\} \tag{3-7}$$

$$\sum_{i=1}^{n} x_i = B_1 \tag{3-8}$$

$$x_i \geqslant 0, \qquad i \in \{1, ..., n\} \tag{3-9}$$

The variable $\lambda \in \Re$ is the respective dual variable of the linear constraint defined in (1-7). Let $X \subset \Re^n$ be the set that satisfies the constraints defined by (3-7), (3-8), and (3-9).

3.3 Optimality Conditions of RAP-LB-L

Most of the basic definitions and properties listed in this section can be found in the book (Rockafellar1993). It is known that the finite intersection of convex sets is convex, thus, we know that the feasible set X is also convex. From Theorem 5.1 stated in (Rockafellar1993), the necessary optimality conditions for the LR of RAP-LB-L are equivalent to the following saddle point conditions:

A couple $(\mathbf{x}^*, \lambda^*)$ is optimal, with $\mathbf{x}^*$ being primal optimal and λ^* being dual optimal, if:

1. The minimum of $\sum_{i=1}^{n} c_i x_i + \lambda^*\Big(\sum_{i=1}^{n} p_i x_i - B_2\Big)$, for $\mathbf{x} \in X$ is attained at $\mathbf{x}^* \in X$.

2. The maximum of $\sum_{i=1}^{n} c_i x_i^* + \lambda\Big(\sum_{i=1}^{n} p_i x_i^* - B_2\Big)$, for $\lambda \in \Re$, is attained at λ^*.

We define the functions g and f, as per (Rockafellar1993):

$$g(\lambda) = \inf_{\mathbf{x}\in X} \sum_{i=1}^{n} c_i x_i + \lambda\Big(\sum_{i=1}^{n} p_i x_i - B_2\Big), \qquad \lambda \in \Re \tag{3-10}$$

$$f(\mathbf{x}) = \sup_{\lambda\in \Re} \sum_{i=1}^{n} c_i x_i + \lambda\Big(\sum_{i=1}^{n} p_i x_i - B_2\Big), \qquad \mathbf{x} \in X \tag{3-11}$$

We denote $W(\lambda_0) \subset X$ to be the set of solutions to $g(\lambda_0)$, i.e. if $\mathbf{w} \in W(\lambda_0)$, then:

$$g(\lambda_0) = \langle \mathbf{c}, \mathbf{w}\rangle + \lambda_0(\langle \mathbf{w}, \mathbf{p}\rangle - B_2) \leqslant \langle \mathbf{c}, \mathbf{x}\rangle + \lambda_0(\langle \mathbf{x}, \mathbf{p}\rangle - B_2), \qquad \forall \mathbf{x} \in X \tag{3-12}$$

Clearly $f(\mathbf{x}) \geqslant L(\mathbf{x}, \lambda) \geqslant g(\lambda)$, for $\mathbf{x} \in X$ and $\lambda \in \Re$. By Corollary 5.4. in (Rockafellar1993), the authors show that the neccessary optimality conditions, for linear programming problems, are also sufficient if the feasible set X forms a polyhedra. Furthemore, if the function g has an optimal value $\lambda^* \in \Re$, then the function f also has an optimal solution $\mathbf{x}^* \in X$, and $g(\lambda^*) = f(\mathbf{x}^*)$.

Given that the feasible set X defined by constraints (1-5), (1-6) and (1-8) are all linear, it follows that X is a polyhedra. Thus a necessary and sufficient optimality condition for the RAP-LB-L is:

If there exists $\mathbf{x}^*$, λ^*, such that $f(\mathbf{x}^*) = g(\lambda^*)$ and $\mathbf{x}^*$ satisfies Equation (1-7), then $(\mathbf{x}^*, \lambda^*)$ is optimal.

PUC-Rio - Certificação Digital Nº 1621891/CA

3.4 Dual Optimal Property

If a function $\varphi : \Re \to \Re$ is concave and differentiable, the derivative at the maximum value λ^* would be equal to zero, i.e. $\frac{\mathrm{d}\varphi}{\mathrm{d}\lambda}(\lambda^*) = 0$. However, in our case, the function g is concave and piecewise-linear. As such, the function is non-differentiable at its breakpoints.

To circumvent this issue, we rely on the notion of subgradient, defined as follows:

Definition 3.1 *A subgradient, v, at x_0, of a concave function $h : \Re \to \Re$ is defined as any value v such that:*

$$h(x) \leqslant h(x_0) + v \cdot (x - x_0) \tag{3-13}$$

The set of subgradients at x_0 is called the subdifferential. We denote the subdifferential at a point x_0 of a function $h : \Re \to \Re$ to be $\partial h(x_o)$. It is implied, from the definition, that the subgradients are not unique at the breakpoints. As such, the subdifferential is an interval. We now state a general property of the optimal dual, λ^* of g.

Property 3.2 *$\lambda^* \in \Re$ is a global maximum of a concave piecewise-linear function g, if and only if, $0 \in \partial g(\lambda^*)$.*

The subdifferentials at λ, in our particular case of function g, are as follows:

$$\partial g(\lambda) = \langle \mathbf{x}, \mathbf{p} \rangle - B_2, \qquad \mathbf{x} \in W(\lambda) \tag{3-14}$$

This can be shown as follows: First we show that it is a valid subgradient, then we show that any subgradient can be obtained through expression (3-14). Let $\mathbf{x} \in W(\lambda)$ be a solution to $g(\lambda)$ and $\mathbf{w}_0 \in W(\lambda_0)$ be solution to $g(\lambda_0)$.

$$g(\lambda) - g(\lambda_0) = \langle \mathbf{c}, \mathbf{x} \rangle + \lambda(\langle \mathbf{p}, \mathbf{x} \rangle - B_2) - g(\lambda_0) \tag{3-15}$$

Substituting Equation (3-15) with Inequality (3-12):

$$\begin{aligned}
&\leqslant \langle \mathbf{c}, \mathbf{w}_0 \rangle + \lambda(\langle \mathbf{p}, \mathbf{w}_0 \rangle - B_2) - g(\lambda_0) && (3\text{-}16)\\
&= \langle \mathbf{c}, \mathbf{w}_0 \rangle + \lambda(\langle \mathbf{p}, \mathbf{w}_0 \rangle - B_2) - \langle \mathbf{c}, \mathbf{w}_0 \rangle - \lambda_0(\langle \mathbf{p}, \mathbf{w}_0 \rangle - B_2) && (3\text{-}17)\\
&= (\langle \mathbf{p}, \mathbf{w}_0 \rangle - B_2)(\lambda - \lambda_0) && (3\text{-}18)
\end{aligned}$$

Finally we conclude, as summarized in Inequality (3-19), that $\langle \mathbf{p}, \mathbf{w}_0 \rangle - B_2$ is a subgradient of $g(\lambda_0)$, where $w_0 \in W(\lambda_0)$.

$$g(\lambda) \leqslant g(\lambda_0) + (\langle \mathbf{p}, \mathbf{w}_0 \rangle - B_2)(\lambda - \lambda_0) \tag{3-19}$$

PUC-Rio - Certificação Digital Nº 1621891/CA

To show that any subgradient of $\partial g(\lambda)$ can be obtained by the expression in (3-14), we first assume that λ is a breakpoint of the piecewise-linear concave function g. If λ is not a breakpoint, then Equation (3-14) is true. Consider two solutions $\mathbf{x}_1, \mathbf{x}_2$, such that $\mathbf{x}_1$ is a solution to one of the linear segments adjacent to the breakpoint at λ and $\mathbf{x}_2$ is a solution to the other adjacent linear segment. Clearly both $\langle \mathbf{x}_1, \mathbf{p} \rangle - B_2$ and $\langle \mathbf{x}_1, \mathbf{p} \rangle - B_2$ are subgradients to g at λ. We claim that the set $W(\lambda)$ is convex. The proof of this claim can be found in Appendix A. Therefore for any $\mu \in (0, 1)$, we have $\mathbf{w} \in W(\lambda)$ where $\mathbf{w} = \mu\mathbf{x}_1 + (1-\mu)\mathbf{x}_2$.

Let $v \in \partial g(\lambda)$. We know that $\langle \mathbf{p}, \mathbf{x}_2 \rangle - B_2 < v < \langle \mathbf{p}, \mathbf{x}_1 \rangle - B_2$. Given that the function $h(\mathbf{x}) = \langle \mathbf{p}, \mathbf{x} \rangle - B_2$ is continuous, and that $W(\lambda)$ is convex, then by the Intermediate Value Theorem, there exists a $\mu \in (0, 1)$ such that $v = \langle \mathbf{x}, \mu\mathbf{x}_1 + (1-\mu)\mathbf{x}_2 \rangle - B_2$. We then conclude that the subgradients of g can be uniquely determined by Equation (3-14).

In case λ^* is a breakpoint, then Property 3.2 implies that there exists a subgradient that is equal to zero, i.e. $\mathbf{x}^* \in W(\lambda^*)$, such that:

$$\langle \mathbf{p}, \mathbf{x}^* \rangle - B_2 = 0 \tag{3-20}$$

The following property guarantees the existence of a breakpoint that is optimal.

Property 3.3 *If g is a piecewise-linear concave function, then there exists at least one breakpoint that is optimal.*

The proof of this well known result can be found in Appendix B.

3.5 Obtaining Primal Feasibility

To complete the chapter, we demonstrate a known method (Ravi1996) on how to obtain a primal feasible solution from a dual optimal value λ^*.

We begin by describing how Algorithm 3.1.1 can be applied to solve g, and how the breakpoints affects the procedure. Given a $\lambda \in \Re$, finding a solution for $g(\lambda)$ is equivalent to solving a RAP-LB. As such, the greedy procedure described in Algorithm 3.1.1, can be used. We rewrite g for better exposition:

$$g(\lambda) = \inf_{x \in X} \sum_{i=1}^{n} (c_i + \lambda p_i)x_i - \lambda B_2 \tag{3-21}$$

At each iteration, the greedy procedure chooses, from the available indices, the weight, $c_i + \lambda p_i$, that is lowest (Line 4). If there are two or more indices with the same weight, i.e. $i \neq j$, $c_i + \lambda p_i = c_j + \lambda p_j$, any way of breaking ties by distributing resources among variables x_i and x_j results in an optimal solution.

PUC-Rio - Certificação Digital Nº 1621891/CA

Such a situation corresponds to λ being a breakpoint, which occurs when there exists $i \neq j$, such that $c_i + \lambda p_i = c_j + \lambda p_j$. For now, assume that there are exactly two variables, x_k and x_l at the breakpoint λ. Given the previous equality we have:

$$g(\lambda) = \inf_{x \in X} \sum_{i=1, i\neq k, i\neq l}^{n} (c_i + \lambda p_i)x_i + (c_k + \lambda p_k)(x_k + x_l) - \lambda B_2 \tag{3-22}$$

At the breakpoint, the solutions, $\mathbf{x}_L$ and $\mathbf{x}_R$, determined by both the left and right adjacent line segments are also valid solutions to $g(\lambda)$. More generally:

Proposition 3.4 *If λ is a breakpoint and $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ are solutions to the left and right lines adjacent to the breakpoint, respectively, then any solution of the form $\boldsymbol{x} = \mu \boldsymbol{x}_L + (1-\mu)\boldsymbol{x}_R$, with $\mu \in (0,1)$ is a solution to g at λ.*

Proof. There are two claims that need to be verified. The first, is that the new solution is part of the feasible set X. The second, is if the new solution obtains the same objective value as $g(\lambda)$.

The first claim is trivial, as X is convex, then it is true by definition. The second claim can be shown to be true as follows:

$$\begin{aligned}
&(\mathbf{c} + \lambda \mathbf{p})(\mu \mathbf{x}_L + (1-\mu)\mathbf{x}_R) - \lambda B_2 && (3\text{-}23)\\
=&\mu((\mathbf{c} + \lambda \mathbf{p})\mathbf{x}_L - \lambda B_2) + (1-\mu)((\mathbf{c} + \lambda \mathbf{p})\mathbf{x}_R - \lambda B_2) && (3\text{-}24)\\
=&\mu g(\lambda) + (1-\mu) g(\lambda) && (3\text{-}25)\\
=&g(\lambda) && (3\text{-}26)
\end{aligned}$$

We have validated that the solution is part of the feasible set X and also that it has the same objective value at $g(\lambda)$ as the other solutions. We conclude that any solution of the form $\mu \mathbf{x}_L + (1-\mu)\mathbf{x}_R$, $\mu \in (0,1)$ is also a solution to g at λ, ■

To complete the demonstration on how to obtain primal feasibility from the dual optimal value, we need to show that there exists a $\mu \in (0,1)$ such that the solution $\mu \mathbf{x}_L + (1-\mu)\mathbf{x}_R$ satisfies the linear constraint (1-7). To begin, we state explicitly, g's subgradient:

$$\frac{\mathrm{d}g}{\mathrm{d}\lambda}(\lambda) = \langle \mathbf{p}, \mathbf{x} \rangle - B_2 \tag{3-27}$$

Given that, $0 \in [\frac{\mathrm{d}g}{\mathrm{d}\lambda}(\lambda_R), \frac{\mathrm{d}g}{\mathrm{d}\lambda}(\lambda_L)]$, and Equation (3-27), it implies the following:

PUC-Rio - Certificação Digital Nº 1621891/CA

$$\langle \mathbf{p}, \mathbf{x}_L \rangle > B_2 \tag{3-28}$$

$$\langle \mathbf{p}, \mathbf{x}_R \rangle < B_2 \tag{3-29}$$

Proposition 3.5 *Let λ^* be an optimal breakpoint and let $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ be solutions to $g(\lambda^*)$, such that equations (3-28) and (3-29) hold. Let $\delta_L > 0$ and $\delta_R > 0$ be such that:*

$$\langle \boldsymbol{p}, \boldsymbol{x}_L \rangle - B_2 = \delta_L \tag{3-30}$$

$$B_2 - \langle \boldsymbol{p}, \boldsymbol{x}_R \rangle = \delta_R \tag{3-31}$$

Then the solution, $\boldsymbol{x}^*$ obtained from the convex sum:

$$\boldsymbol{x}^* = \mu \boldsymbol{x}_L + (1 - \mu)\boldsymbol{x}_R \tag{3-32}$$

Where $\mu = \frac{\delta_R}{\delta_L + \delta_R}$, is a primal optimal solution.

Proof.

From Proposition 3.4, we know that any convex sum obtains the same objective function and satisfy the same constraints. We are left to show that this new solution, $\mathbf{x}^*$ satisfies constraint (1-7), the linear constraint.

$$\begin{aligned}
\langle \mathbf{p}, \mathbf{x}^* \rangle &= \langle \mathbf{p}, \mu \mathbf{x}_L + (1 - \mu)\mathbf{x}_R \rangle && (3\text{-}33)\\
&= \mu \langle \mathbf{p}, \mathbf{x}_L \rangle + (1 - \mu)\langle \mathbf{p}, \mathbf{x}_R \rangle && (3\text{-}34)\\
&= \mu(B_2 + \delta_L) + (1 - \mu)(B_2 - \delta_R) && (3\text{-}35)\\
&= B_2 - \delta_R + \mu(\delta_L + \delta_R) && (3\text{-}36)\\
&= B_2 - \delta_R + \frac{\delta_R}{\delta_L + \delta_R}(\delta_L + \delta_R) && (3\text{-}37)\\
&= B_2 && (3\text{-}38)
\end{aligned}$$

We've found a solution that satisfies all of the original problem's constraints, while maintaining g's objective value. From the saddle point condition we have, $F(\mathbf{x}) = L(\mathbf{x}, \lambda) = g(\lambda)$, and thus $\mathbf{x}^*$ is an optimal solution. ■

3.6 Problem Feasibility Check

Using the following feasibility check it is possible to determine an interval Λ that includes all of g's linear segments. It can be used as a set up for the weakly polynomial algorithm. The procedure has complexity $O(n \log n)$ and so it does not alter the overall complexity of the weakly polynomial algorithm.

PUC-Rio - Certificação Digital Nº 1621891/CA

Consider the associated weights of each variable as lines, $y_i(\lambda) = c_i + \lambda p_i$. For $\lambda \to -\infty$, the order, π, of $y_i(\lambda)$ is equivalent to the order of p_i. Analogously, for $\lambda \to \infty$, the order of $y_i(\lambda)$, is equivalent to the inverse order, π^{-1}, of p_i. In both orders, the elements, $y_i(\lambda)$, have the same neighbours, i.e. if $y_{\pi(i)}(\lambda)$ is adjacent to $y_{\pi(i+1)}(\lambda)$ in order π, then $y_{\pi^{-1}(i)}(\lambda)$ is also adjacent to $y_{\pi^{-1}(i+1)}(\lambda)$. Each intersection of lines y_i and y_j is equivalent to an adjacent swap in the order π. Therefore, the first and last intersections, λ_F and λ_L, are equivalent to the first and last adjacent swaps.

Let k and l be indices such that at $y_{\pi(k)}(\lambda_F) = y_{\pi(k+1)}(\lambda_F)$ and $y_{\pi(l)}(\lambda_L) = y_{\pi(l+1)}(\lambda_L)$. To determine k and l, we sort the variables according to p_i, to obtain the order π. Afterwards, the intersections $\lambda_{\pi(i),\pi(i+1)}$ is calculated for $i \in \{1, ..., n-1\}$. Then, we determine the minimum and maximum values, $\lambda_{\min}$ and $\lambda_{\max}$. The range $[\lambda_{\min}, \lambda_{\max}]$ includes all intersections of the lines y_i, and thus all linear segments of g.

Problem feasibility is determined by using Algorithm 3.1.1 to solve both $g(\lambda_{\min})$ and $g(\lambda_{\max})$, in order to obtain the subgradients at $\lambda_{\min}$ and $\lambda_{\max}$. If the subgradients have opposite signs, then by the Intermediate Value Theorem, there is a λ^*, with $\lambda_{\min} < \lambda^* < \lambda_{\max}$ where there exists a subgradient of $g(\lambda^*)$ that is zero. We conclude that the problem is feasible if and only if the subgradients have opposite signs.

The complexity of the overall feasibility check is $O(n \log n)$, due to the sorting algorithm. Furthermore, by defining $\Lambda = [\lambda_{\min}, \lambda_{\max}]$, we guarantee to always find a solution for the weakly polynomial algorithm, if it exists.

PUC-Rio - Certificação Digital Nº 1621891/CA

4
Weakly Polynomial Algorithm

In this Chapter, assume that the RAP-LB-L is well-behaved as follows. For any λ_0 such that the costs of two variables are equal, $c_i + \lambda_0 p_i = c_j + \lambda_0 p_j$, for $i \neq j$, $i, j \in \{1, ..., n\}$, then:

1. There is no $k \in \{1, ..., n\}$, $k \neq i$, $k \neq j$, such that $c_k + \lambda_0 p_k = c_i + \lambda_0 p_i$.
2. There are also no $l, k \in \{1, ..., i-1, i+1, ..., j-1, j+1, ..., n\}$, $l \neq k$, such that $c_k + \lambda_0 p_k = c_l + \lambda_0 p_l$

If we consider the costs to be lines on a plane, then the above two assumptions are equivalent to having each intersection, among the set of lines, to occur at unique λ values. An intersection of two lines, is thus, uniquely characterize by a value λ. If these assumptions are not fulfilled a small pertubation of the parameters can be applied.

The proposed weakly polynomial algorithm, denoted as WeaklyPoly, performs a binary search over the dual space, with the aim of finding a solution that produces a subgradient of zero. The initial search interval, Λ, is assumed to be closed and large enough to contain all breakpoints of the function g. As per binary search, at each iteration, the procedure reduces the search interval by half. The choice of which half to discard is based on the subgradient attained at the mid point, λ_M. Considering that in a concave function, such as g, the gradients are non-decreasing with respect to λ, by obtaining the subgradient at λ_M we can update the interval Λ accordingly. Thus, if the subgradient at the mid-point is positive, we discard the lower half of Λ. Conversely, if the subgradient at the mid-point is negative, we discard the upper half of Λ.

The setup phase requires to find the order of the weights $y_i(\lambda) = c_i + \lambda p_i$, at both ends of the search interval. At each iteration, the two orders of the weights y_i are determined, one at each end of the search interval Λ. By counting the inversions between the two orders, we can develop a stopping criteria for our binary search. Given that our data is well-behaved, we guarantee that a breakpoint is uniquely characterized by one intersection. Thus, whenever the count inversions between the two orders is one, we are sure to have only one breakpoint in the search interval. Given that the solution found on the lower end of the search interval results in a positive subgradient, and the solution

PUC-Rio - Certificação Digital Nº 1621891/CA

found on the higher end of the search interval results in a negative subgradient, we can perform a convex sum to obtain a primal feasible solution. The solution is then optimal.

We show the algorithm in detail next.

Algorithm 4.0.1: WeaklyPoly

```
1  Λ ← FeasibilityCheck();
2  if Λ = ∅ then
3  |  return ∅ ;
4  end
5  λ_L ← min{Λ};
6  λ_R ← max{Λ};
7  while CountInversion(λ_L, λ_R) > 1 do
8  |  λ_M ← (λ_L + λ_R)/2;
9  |  x ← Greedy(λ_M);
10 |  v_M ← Σ_{i=1}^{n} p_i x_i − B_2;
11 |  if v_M > 0 then
12 |  |  λ_L ← λ_M;
13 |  else if v_M < 0 then
14 |  |  λ_R ← λ_M;
15 |  else
16 |  |  return x;
17 |  end
18 end
19 x_L ← Greedy(λ_L);
20 x_R ← Greedy(λ_R);
21 x* ← ConvexSum(x_L, x_R);
22 return x*;
```

4.1 Correctness and Complexity

The first six lines are part of the setup phase, where the search space Λ is initiated using the procedure described in Section 3.6. If the problem is infeasible, we return the empty set, Lines 2 through 4. In Lines 5 and 6, endpoints of Λ are determined. The *CountInversion* procedure in Line 7 evaluates the weights at λ_L and λ_R, i.e. calculates the values $y_i(\lambda) = c_i + \lambda p_i$. It then proceeds to sort them in increasing order, producing two orders, on at λ_L and another at λ_R. The procedure then counts the inversions between the two lists, and returns the number of inversions.

PUC-Rio - Certificação Digital Nº 1621891/CA

If the problem is trivial, i.e. has zero or one total inversion, then the loop defined in Lines 7 to 18 is not executed. If there are zero inversions between the weights evaluated at λ_L and the ones evaluated at λ_R, then necessarily, the problem is trivial (we assume feasibility), and the lines 15 and 16 will both return the optimal solution $\mathbf{x}^*$. The $ConvexSum$ procedure performs a convex sum, which will also return the optimal solution.

If there is only one inversion, it implies that there is only one intersection in the search space. If this intersection does not correspond to a breakpoint, then the problem is also trivial, as, assuming the problem to be feasible, this would imply that any solution of $g(\lambda)$, with $\lambda \in \Lambda$ is optimal. The loop in Line 7 is not executed, and Lines 19 through 21 return the optimal solution. If the intersection is a breakpoint, then the solutions $\mathbf{x}_L$ and $\mathbf{x}_R$ found in Lines 19 and 20 will not be equal. The procedure $ConvexSum$ in Line 21, is defined by Proposition 3.5. $ConvexSum$ has as inputs solutions $\mathbf{x}_L$ and $\mathbf{x}_R$, and results in solution $\mathbf{x}^*$. From Proposition 3.5, we know that $\mathbf{x}^*$ is primal optimal.

If, on the other hand the problem is not trivial, then $CountInversion(\lambda_L, \lambda_R) > 1$, and the loop in Line 7 is executed. Line 8 finds the midpoint, λ_M, of the search interval, Λ. The solution to $g(\lambda_M)$, $\mathbf{x}$, is obtained in Line 9. We obtain a subgradient in Line 10. We update the search interval in Lines 11 through 17. If the subgradient v_M is positive (Line 11), we discard the left half (Line 12). Analogously, we update the search interval accordingly, if the subgradient is negative (Lines 13 and 14). If the subgradient, v_M is zero, then by Property 3.2, $\mathbf{x}$ is primal optimal, and Line 16, returns the optimal solution.

At each iteration Algorithm 4.0.1 guarantees that $g(\lambda_L)$ will always have a positive subgradient, and $g(\lambda_R)$ will always have a negative subgradient. The iterations stops if either one has zero subgradient and returns the optimal solution. If the loop terminates without returning an optimal solution, then it implies there is only one intersection (Line 7). The opposing signs of $g(\lambda_L)$ and $g(\lambda_R)$ subgradients guarantees that the intersection is a breakpoint. Thus, by Proposition 3.5, the convex sum of the solutions at the endpoints, $\mathbf{x}_L$ and $\mathbf{x}_R$ is primal optimal.

The complexity of the while loop is as follows. The while loop runs a binary search on the interval Λ. In the worst case, the loop terminates when $CountInversions(\lambda_L, \lambda_R) > 1$. This only happens when there is only one intersection in the interval (λ_L, λ_R). Let us denote I_k as the interval where each linear segment of g is defined on, and let I be the smallest one, nameley $|I| \leqslant |I_k|$. Then, whenever the search interval becomes smaller than I, the loop terminantes, as there necessarily cannot be another breakpoint in the search

PUC-Rio - Certificação Digital Nº 1621891/CA

interval. Therefore, the loop iterates a total of $O(\log \frac{|\Lambda|}{|I|})$ time.

In each iteration the greedy and count inversions procedures are executed once. As shown in Algorithm 3.1.1, the greedy procedure takes $O(n)$ time. Calculating the subgradient can be performed in linear time. It is known that the count inversions algorithm can be implemented in $O(n \log n)$ time. Therefore, each iteration of the while loop, we perform $O(n \log n)$ steps. Combining with the iterations of the while loop, we obtain the complexity $O(n \log n \log \frac{|\Lambda|}{|I|})$ for our algorithm.

PUC-Rio - Certificação Digital Nº 1621891/CA

5
Strongly Polynomial Algorithm

In this Chapter we first describe the general framework for the parametric search using the example first exposed in (Megiddo1983). We then show how to apply the general framework to our own problem. Finally we discuss the complexity and prove its correctness.

5.1
Parametric Search

Parametric search is a technique to design efficient sequential algorithms from parallel algorithms. The parallel algorithm is simulated on a sequential machine and efficiency is gained by optimizing the simulation, we refer to (Megiddo1983), (Cole1987), and (Goodrich2013) for additional explanations.

Given a decision problem $\mathcal{B}$ that we wish to solve, there are three requirements that need to be met for the technique to be applicable. We denote these requirements as Parametric Search Requirements (PSR).

1. The decision problem $\mathcal{B}$ must depend monotonically on a parameter λ, such that it is true in an interval $(-\infty, \lambda^*]$ and false every where else.

2. There must exist a sequential algorithm $\mathcal{C}$ that can determine whether if, for a given λ, it is less than, equal to, or greater than λ^*.

3. A choice of a generic, comparison based, parallel algorithm $\mathcal{A}$ that is independent of the parameter λ.

The goal of the parametric search is to find the largest λ such that $\mathcal{B}$ is true, i.e. find λ^*. The idea is to use the parallel algorithm $\mathcal{A}$ that runs in time T on a machine with P processors, to design a serial algorithm that solves the problem $\mathcal{B}$ in time $T \log P$.

To achieve this, the parallel algorithm $\mathcal{A}$ is executed on the unknown value λ^*, either as its input, or on inputs that depend on λ^*. Since $\mathcal{A}$ is comparison based, whenever a comparison needs to be resolved, it relies on $\mathcal{C}$ to solve it. The result of this scheme is that the value λ^* is found as a by-product. Once found, $\mathcal{B}$ is solved.

PUC-Rio - Certificação Digital Nº 1621891/CA

Note that the problem that algorithm $\mathcal{A}$ solves is not decision problem $\mathcal{B}$. The sequential algorithm $\mathcal{C}$ is used to bridge the gap between the problem that algorithm $\mathcal{A}$ solves and the decision problem $\mathcal{B}$. This way, algorithm $\mathcal{C}$ ties the execution of algorithm $\mathcal{A}$ to solving problem $\mathcal{B}$.

Consider the following example from (Megiddo1983): There are n functions f_i of the form $f_i(\lambda) = a_i + \lambda b_i$, $i \in [n]$, such that the functions f_i are pairwise distinct and all b_i's are positive. Let $F(\lambda)$ be a function such that $F(\lambda) = median\{f_1(\lambda), f_2(\lambda), ..., f_n(\lambda)\}$. It is clear that $F(\lambda)$ can be solved in $O(n)$ time with the median-finding algorithm and is a monotonic increasing, piecewise linear function with $O(n^2)$ breakpoints. The goal is to find λ, such that $F(\lambda) = 0$. We denote λ^* such that $F(\lambda^*) = 0$.

A straight forward algorithm can be implemented that enumerates all $O(n^2)$ breakpoints and performs a binary search. At each iteration of the binary search, the median breakpoint, i.e. the intersection λ_{ij} of two lines $f_i(\lambda)$ and $f_j(\lambda)$ is selected, and $F(\lambda_{ij})$ is solved. If $F(\lambda_{ij}) < 0$ then $\lambda_{ij} < \lambda^*$, likewise if $F(\lambda_{ij}) > 0$ then $\lambda_{ij} > \lambda^*$. At the end of the binary search, the search interval will be $(\lambda_{st}, \lambda_{kl})$. Since there are no breakpoints in the final search interval, $F(\lambda)$ is linear in $(\lambda_{st}, \lambda_{kl})$, and the solution to $F(\lambda) = 0$ can be found in one step. This algorithm takes $O(n^2)$ time to enumerate all of the breakpoints, and then $O(\log n)$ iterations of solving $F(\lambda)$ resulting in a total complexity of $O(n^2)$.

In order to improve the complexity, we employ the parametric search technique. The first PSR can be satisfied by considering the decision problem $\mathcal{B}$ to be: "Is λ such that $F(\lambda) \leqslant 0$?". The second PSR is satisfied by using the following procedure as algorithm $\mathcal{C}$: For a given λ, solve $F(\lambda)$ and looks at its sign, if $F(\lambda) < 0$ return $\lambda < \lambda^*$, if $F(\lambda) = 0$ return $\lambda = \lambda^*$, and finally, if $F(\lambda) > 0$ return $\lambda > \lambda^*$. The third PSR is satisfied by using the parallel sorting algorithm AKS (Ajtai1983). The AKS algorithm uses $P = O(n)$ processors at each time step and takes a total of $T = O(\log n)$ time. Each processor is used to resolve a comparison between two functions. The inputs to algorithm $\mathcal{A}$ are the functions $\{f_1(\lambda^*), f_2(\lambda^*), ..., f_n(\lambda^*)\}$ evaluated at the unknown value λ^*.

Whenever algorithm $\mathcal{A}$ compares two functions f_i and f_j, the equation $f_i(\lambda) = f_j(\lambda)$ is solved which finds the breakpoint λ_{ij}. Thus after one time step of the parallel algorithm, we have P breakpoints, which were all produced independently of each other. We do not assume that the P breakpoints are sorted. Let these values be $\lambda_1 \leqslant \lambda_2 \leqslant ... \leqslant \lambda_P$. We optimize the simulation of $\mathcal{A}$ by using algorithm $\mathcal{C}$ to perform a binary search on the P breakpoints. The end result is a reduction of our search interval to $(\lambda_i, \lambda_{i+1})$ for some $i \in [P]$, such that $\lambda^* \in (\lambda_i, \lambda_{i+1})$. To summarize, one iteration of the whole

PUC-Rio - Certificação Digital Nº 1621891/CA

algorithm includes simulating one parallel step of algorithm $\mathcal{A}$ producing P breakpoints followed by using $\mathcal{C}$ to perform a binary search to completion on the P breakpoints. After all iterations of $\mathcal{A}$ are executed, we are guaranteed to have an interval, $(\lambda_k, \lambda_{k+1})$, for some $k \in [O(n^2)]$ where $F(\lambda)$ is linear in the interval. The solution to $F(\lambda) = 0$ can then be found in constant time.

Algorithm $\mathcal{C}$ runs in $O(n)$ time, and the binary search has a total of $O(\log P)$ iterations, therefore the binary search takes $O(n \log P)$ time. The sorting algorithm has $O(\log n)$ steps and thus the total complexity of the algorithm is $O(n \log P \log n)$, a significant improvement from $O(n^2)$.

5.2 Parametric Search Applied to RAP-LB-L

As described in Chapter 3, the breakpoints happen whenever the weight associated to two variables x_i and x_j are equal, i.e. $c_i + \lambda p_i = c_j + \lambda p_j$. If we consider each weight to define a line on the dual space, the breakpoints occur at the intersection of the lines.

It is important though, to note that not all intersections are breakpoints. Consider the LR of the following RAP-LB-L, with $B_2 = 0$, no nested constraints and $B_1 = 10$:

$$\min_{\mathbf{x}} (5 - \lambda)x_1 + (5 + 2\lambda)x_2 + (-2 + 6\lambda)x_3 \tag{5-1}$$

Figure 5.1 illustrates the example. The intersections of the lines, $y_1 = 5 - \lambda$, $y_2 = 5 + 2\lambda$, and $y_3 = -2 + 6\lambda$, occur at values $\lambda_1 = 0$, $\lambda_2 = 1$ and $\lambda_3 = 2$. Lines y_1 and y_2 intersect at $\lambda = \lambda_1$, however, it does not affect the value of the objective function, as all resource is allocated to variable x_3. At the following intersection, λ_2, the resource is allocated to x_2 and x_3 is set to zero.

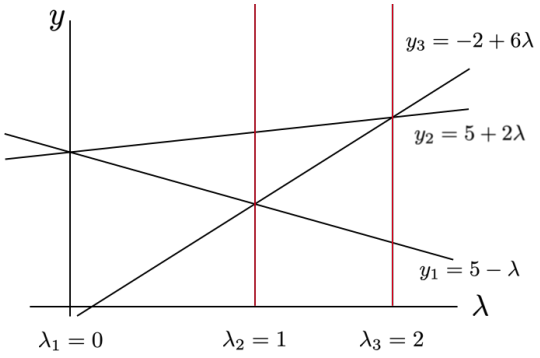


Figure 5.1: Lines y_1, y_2, and y_3 and their intersections at λ_1, λ_2, λ_3.

In this regard, the LR of RAP-LB-L does not necessarily have only one intersection that is dual optimal.

PUC-Rio - Certificação Digital Nº 1621891/CA

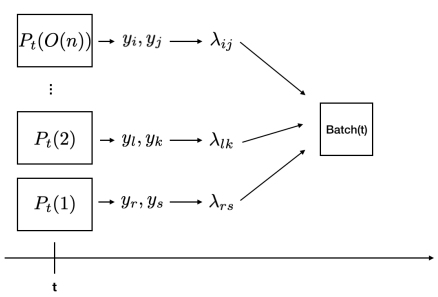
We assume that there does not exist a i and j, such that $c_i = c_j$ and $p_i = p_j$. If it does, we remove the variable belonging to the smallest of the nested sets, i.e. if $i \in J_1$ and $j \in J_2$ such that $J_1 \subset J_2$, then we remove x_i. This does not affect the problem as any resource allocated to x_i can be absorbed by x_j without affecting the objective function or the subgradient. After solving to optimality, any allocation between x_i and x_j is trivially optimal as well.

Applying the parametric search technique to our problem requires to first, satisfy the PSR. The first PSR is satisfied by considering the following decision problem $\mathcal{B}$: "Is the subgradient of g evaluated at λ greater or equal to zero?", i.e. deciding whether $\partial g(\lambda) \geqslant 0$ is true or not. The second PSR is satisfied by choosing the AKS sorting network as algorithm $\mathcal{A}$. For now, we will assume that there exists an algorithm $\mathcal{C}$ that satisfies the third PSR. The details of algorithm $\mathcal{C}$ will be postponed to the next section.

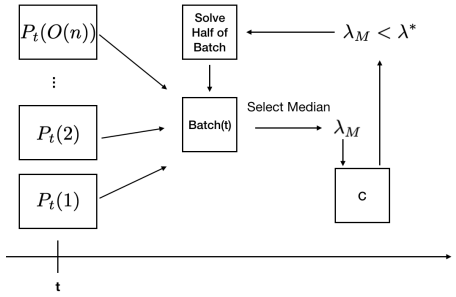
Similar to what was described in Section 5.1, the execution of the overall algorithm is as follows: We simulate the execution of the AKS algorithm on the inputs $\{y_1(\lambda^*), y_2(\lambda^*), ..., y_n(\lambda^*)\}$ on a serial machine. At each parallel step, each processor of the AKS compares two values $y_i(\lambda^*)$ and $y_j(\lambda^*)$ which results in finding the intersection λ_{ij} such that $y_i(\lambda_{ij}) = y_j(\lambda_{ij})$. As such, each simulated parallel step of the AKS algorithm results in P intersections. A binary search is then employed, using the median-finding algorithm to select λ_M, followed by the execution of algorithm $\mathcal{C}$ to determine whether $\lambda_M < \lambda^*$, $\lambda_M > \lambda^*$, or $\lambda_M = \lambda^*$. By the same reasoning used before, we're able to reduce the search interval after each iteration of the AKS sorting network.

Figure 5.2a illustrates the comparisons made by each processor at iteration t of the AKS sorting network. The figure shows that each processor compares different pairs of lines. Each processor finds the intersection of the two lines they are comparing and adds them to the batch. In figure 5.2b, illustrates the binary search process over the set of values produced by the processors.

PUC-Rio - Certificação Digital Nº 1621891/CA



(a) Each processor produces a value and adds it to a batch.



(b) Binary search process over the produced values.

Figure 5.2: Illustration of each step of the simulation of one parallel step of AKS sorting network.

If $C(n)$ is the running time of algorithm $\mathcal{C}$, then the binary search runs in $O(C(n) \log P)$ time. One simulated iteration of the AKS sorting network takes $O(P + C(n) \log P)$ time, where $O(P)$ is the time taken to simulate the comparisons made by all processors. Given than the AKS sorting network uses $P = O(n)$ processors at each iteration and runs a total of $O(\log n)$ iterations, the overall complexity of the algorithm is $O((n + C(n) \log n) \log n)$.

5.3 Sequential Algorithm $\mathcal{C}$

The sequential algorithm $\mathcal{C}$ finds two solutions, $\mathbf{x}_L$ and $\mathbf{x}_R$, for g at intersection λ_{ij} of the lines $l_i : y_i = c_i + \lambda p_i$ and $l_j : y_j = c_j + \lambda p_j$. The solutions are such, that the generated subgradients are the smallest and largest. If the subgradients obtained from $\mathbf{x}_L$ and $\mathbf{x}_R$ have opposite signs, then a third solution, $\mathbf{x}^*$, to $g(\lambda_{ij})$ is obtained through the convex sum detailed in Proposition 3.5. The obtained solution $\mathbf{x}^*$ is then optimal and $y_i(\lambda^*) = y_j(\lambda^*)$ is returned. If the subgradients obtained from $\mathbf{x}_L$ and $\mathbf{x}_R$ have the same sign, then the subgradient of $\mathbf{x}_L$ is used to determine whether $y_i(\lambda^*) > y_j(\lambda^*)$ or $y_i(\lambda^*) < y_j(\lambda^*)$. It is worth mentioning that in this case, if intersection λ_{ij} is not a breakpoint, then the subgradients, v_L and v_R are equal, and both solutions are equal to each other.

Next, we illustrate algorithm $\mathcal{C}$ in detail. We assume w.l.o.g. that the gradients of the lines l_i and l_j are such that $p_i > p_j$.

PUC-Rio - Certificação Digital Nº 1621891/CA

Algorithm 5.3.1: $\mathcal{C}(l_i, l_j)$

```
1  λ_ij ← intersection of lines l_i and l_j;
2  x_L, x_R ← ModifiedGreedy(λ_ij) ;
3  v_L ← ⟨p, x_L⟩ − B_2;
4  v_R ← ⟨p, x_R⟩ − B_2;
5  if v_L × v_R ⩽ 0 then
6  |  x* ← ConvexSum(x_L, x_R) ;
7  |  return y_i(λ*) = y_j(λ*);
8  end
9  if v_L < 0 then
10 |  return y_i(λ*) > y_j(λ*);
11 end
12 if v_L > 0 then
13 |  return y_i(λ*) < y_j(λ*);
14 end
```

PUC-Rio - Certificação Digital Nº 1621891/CA

As described before, Line 2 returns two solutions from $W(\lambda_{ij})$, who's subgradients at $g(\lambda_{ij})$ are the maximum and minimum possible. In Lines 3 and 4, the subgradient with respect to each solution is calculated. In Line 5 we check whether the subgradients have the same sign or not by checking their product. If the product is negative, then they have different signs which implies that $0 \in \partial g(\lambda_{ij})$. If the product is zero, then either one or both subgradients is zero. Either case, the convex sum returns us a third solution (Line 6) $\mathbf{x}^*$ at $g(\lambda_{ij})$ as shown in Proposition 3.5. The convex sum, guarantees us that the subgradient of $\mathbf{x}^*$ at $g(\lambda_{ij})$ is zero, and thus by Proposition 3.5, $\mathbf{x}^*$ is optimal, and $\mathcal{C}$ returns that $y_i(\lambda^*) = y_j(\lambda^*)$. If, on the other hand, the product is positive, we're not at the dual optimal. We then use the subgradient v_L to determine whether Algorithm $\mathcal{C}$ returns $y_i(\lambda^*) > y(\lambda^*)$ (if $v_L < 0$, Line 9) or $y_i(\lambda^*) < y(\lambda^*)$ (if $v_L > 0$, Line 12).

Before discussing the complexity of $\mathcal{C}$, we show how the modified greedy procedure finds the solutions.

Algorithm 5.3.2: ModifiedGreedy(λ)

1 $S_{m+1} \leftarrow \varnothing$;
2 $J_0 \leftarrow \varnothing$;
3 $\mathbf{x}_L \leftarrow 0$;
4 $\mathbf{x}_R \leftarrow 0$;
5 **for** $k \leftarrow m$ **to** 1 **do**
6 $\quad \Delta \leftarrow b_k - b_{k-1}$;
7 $\quad S_k \leftarrow \arg\min_i \{c_i + \lambda p_i | \, i \in (J_k - J_{k-1}) \cup S_{k+1}\}$;
8 $\quad i_L \leftarrow \arg\max_i \{p_i | \, i \in S_k\}$;
9 $\quad i_R \leftarrow \arg\min_i \{p_i | \, i \in S_k\}$;
10 $\quad x_{i_L} \leftarrow x_{i_L} + \Delta$;
11 $\quad x_{i_R} \leftarrow x_{i_R} + \Delta$;
12 $\quad S_k \leftarrow \{i_L, i_R\}$;
13 **end**

PUC-Rio - Certificação Digital Nº 1621891/CA

The values b_k are the upper nested constraints, where $b_m = B_1$. Given our assumption that $b_k > b_{k-1}$, the allocation of resources to variables in J_{k-1} is limited to b_{k-1}. Thus, the amount $b_k - b_{k-1}$ of resource can only be allocated to variables in J_m that are not in J_{k-1}. In each iteration, we retain the indices of minimum weight of $J_m - J_{k-1}$ (Line 5). Lines 6 and 7 chooses the index that would have a lower cost, if λ was marginally lower (Line 6) and marginally greater (Line 7) than λ_{ij}, even though at the intersection λ_{ij}, the weights are equal, $c_i + \lambda p_i = c_j + \lambda p_j$. However, if $\lambda < \lambda_{ij}$, and $p_i > p_j$, then $c_i + \lambda p_i < c_j + \lambda p_j$. The order of the lines before and after an intersection is uniquely determined by the order of their gradients, namely p_i and p_j. The situation is analagous for $\lambda > \lambda_{ij}$. Lines 8 and 9 allocate resource to the solutions appropriately. We update the minimum variables in Line 10 to prepare for the next iteration.

If, on the other hand, the intersection λ_{ij} is not a breakpoint, then both solutions $\mathbf{x}_L$ and $\mathbf{x}_R$ are equal.

The modified greedy procedure finds two solutions to $g(\lambda)$ in $O(n)$ time. The minimum of a set K can be determined in $O(|K|)$. At each iteration of the loop, we determine the minimum element of the set $J_k - J_{k-1}$, for a $k \in \{1, ..., m\}$. The collection of sets $J_k - J_{k-1}$ form a partition of the set $J_m = \{1, ..., n\}$. As such, the procedure iterates over the set J_m, resulting in a total of $O(n)$ operations.

5.4 Complexity and Improvement

Clearly, Lines 3, 4 and 6 take $O(n)$ time. Coupled with the linear complexity of the modified greedy procedure, we conclude that our algorithm $\mathcal{C}$ takes $O(n)$ time. Therefore the overall complexity of the strongly polynomial algorithm is $O(n \log^2 n)$. This result can be improved by utilizing (Cole1987)'s improvement of the parametric search. The improvement is specific to the cases of parametric search where the problem that algorithm $\mathcal{A}$ solves is sorting.

In the paper, Cole used the AKS sorting network to show the results. The key insight made by Cole is the observation of the inefficiency of running a binary search to completion at each iteration of the simulated parallel step of algorithm $\mathcal{A}$. The framework of Megiddo has a clear distinction of the two processes involved in the overall algorithm. The first process is the simulation of a parallel step of algorithm $\mathcal{A}$. The second process is running the binary search to completion on the values given by the first process.a Megiddo's parametric search can then be seen as successive alternating between process one and process two.

In Cole's improvement, however, these two processes are intertwined. Cole observed that after each iteration of the binary search, new comparisons are enabled. Comparisons, that will only appear in later batches. Therefore, after each iteration of the binary search, the newly enabled comparisons are added to the current batch, before continuing on with the binary search. This way, the binary search is able to operate over multiple batches before terminating.

PUC-Rio - Certificação Digital Nº 1621891/CA

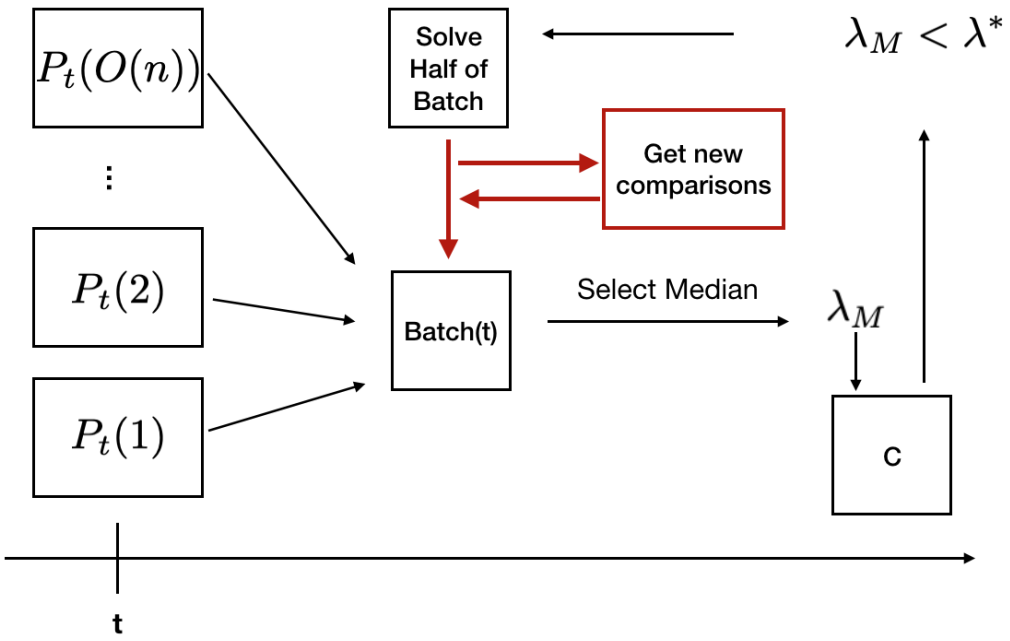


Figure 5.3: Cole's improvement

Figure 5.3 illustrates the updated algorithm with Cole's improvement. With Cole's improvement, the first iteration of algorithm $\mathcal{A}$, where the comparison values λ are produced, is simulated as previously described, without alterations. The binary search is then iterated once. This iteration includes, selecting the median and using it as input to $\mathcal{C}$, and then solving half of the batch. Before proceeding to the next iteration, however, Cole's improvement gets new comparisons that were enabled and add the corresponding values to the current batch.

Technical details of Cole's improvement were omitted in order to simplify the exposition. We refer to (Cole1987) for a complete description of the improvement.

Cole showed that with this modification, a factor of $O(\log n)$ can be removed from the overall complexity. This allows us an improvement to $O(n \log n)$ from $O(n \log^2 n)$, as desired.

PUC-Rio - Certificação Digital Nº 1621891/CA

6
Implementation

We ran computational experiments to compare the performance of the proposed algorithms to the IP method. The choice of the IP method as a a benchmark is due to the lack of a dedicated algorithm to solve the RAP-LB-L. The implementation of the weakly polynomial algorithm is the same as defined in Chapter 4.

The AKS sorting network is required for the strongly polynomial algorithm. However, its implementation is complicated and, furthermore, it also has a large constant factor in its computational complexity. For these reasons, we chose to use a different sorting algorithm for the strongly polynomial algorithm implementation. We followed the work done by (vanOostrum2004). The authors use quicksort instead of the AKS sorting network, choosing the first element of the array as the pivot element. Quicksort's average complexity of $O(n \log n)$ makes this implementation practical, albeit at a worse theoretical complexity. The theoretical worse-case complexity of the strongly polynomial implementation is therefore $O(n^2 \log n)$, with an average complexity of $O(n \log^2 n)$.

It is not possible to attain an average complexity of $O(n \log n)$ with quicksort, as seen in (vanOostrum2004). This is due to the inability of applying the optimization of (Cole1987) when using quicksort. In quicksort, the batch of comparisons are the comparisons of each element of the set to the pivot. In order to obtain the next batch, a new pivot needs to be chosen. The choice of the new pivot is only possible after partitioning the elements into two sets, one set where every element is less than the pivot, and another set where every element is greater than the pivot. Thus, all comparisons of one batch needs to be resolved before proceeding to the next batch. This makes Cole's optimization not possible to implement.

However, (Goodrich2013) developed a parametric search framework that has the same complexity as (Cole1987) with high probability. The AKS sorting network is substituted with the box sort, and the weight assignment is modified to accomodate the changes. Due to the complexity of managing the weights of different comparisons, we implemented the quicksort as algorithm $\mathcal{A}$.

Before running the weakly polynomial algorithm on the problem in-

PUC-Rio - Certificação Digital Nº 1621891/CA

stances, the $O(n \log n)$ feasibility check procedure was used to calculate the interval Λ.

6.1 Problem Instances

The algorithms were implemented in C++ and run on a single core of Intel i5 2.6GHz CPU. For problem instances where a run took less than one second, they were repeated a total of 100 times. For runs that took less than a minute but longer than a second, they were repeated a total of 10 times. For runs that took longer than a minute, they were only run once. The average time was determined for each problem instance. The number of variables of the instances are $n \in \{10, 20, 50, 100, 200, 500, 1000, ..., 10^6\}$. For each problem size, 10 different instances were randomly generated. Every problem instance has the same number of nested constraints as decision variables. The experiments were run on three different sets of problem instances: Uniform, Increasing, and Decreasing. The first, are the original ones used in (Vidal2018), with extended parameters.

We refer to (Vidal2018) for a more detailed explanation on the generation of parameters. The weights c_i were generated randomly in the range $[0, 1]$. The parameters l_i and u_i, generated in (Vidal2018), were used to generate the upper nested constraints as follows: First, a sequence of values v_i and w_i were generated, where $v_0 = w_0 = 0$, $v_i = v_{i-1} + X_i^v$, and $w_i = w_{i-1} + X_i^w$. The parameters X_i^v and X_i^w were generated from a uniform distribution, $U(l_i, u_i)$ for $i \in \{1, ..., n\}$. Finally, the sequence is used to obtain the upper nested constraint, $b_i = \max\{v_i, w_i\}$.

For the Decreasing and Increasing problem set, the weights c_i were generated in the range $[0, 1]$ and a parameter α_i was uniformly generated in the range $[0, 0.5]$, for $i \in \{1, ..., n\}$. For the Increasing problem set, the parameters α_i were sorted in increasing order and for the Decreasing problem set, the paramters α_i were sorted in decreasing order. For both problem sets, the upper nested constraints were obtained as follows: $b_i = b_{i-1} + \alpha_i$, where we define $b_0 = 0$.

The parameters for the linear constraints, p_i, were randomly generated from a uniform distribution in $[-1, 1]$. Afterwards, the parameter B_2 was randomly generated from a uniform distribution in the range $[B_1 \times \min\{p_i\}, B_1 \times \max\{p_i\}]$. This however does not guarantee problem feasibility. In case the resulting problem is infeasible, according to the feasibility check described in Section 3.6, B_2 is regenerated.

PUC-Rio - Certificação Digital Nº 1621891/CA

6.2 Results

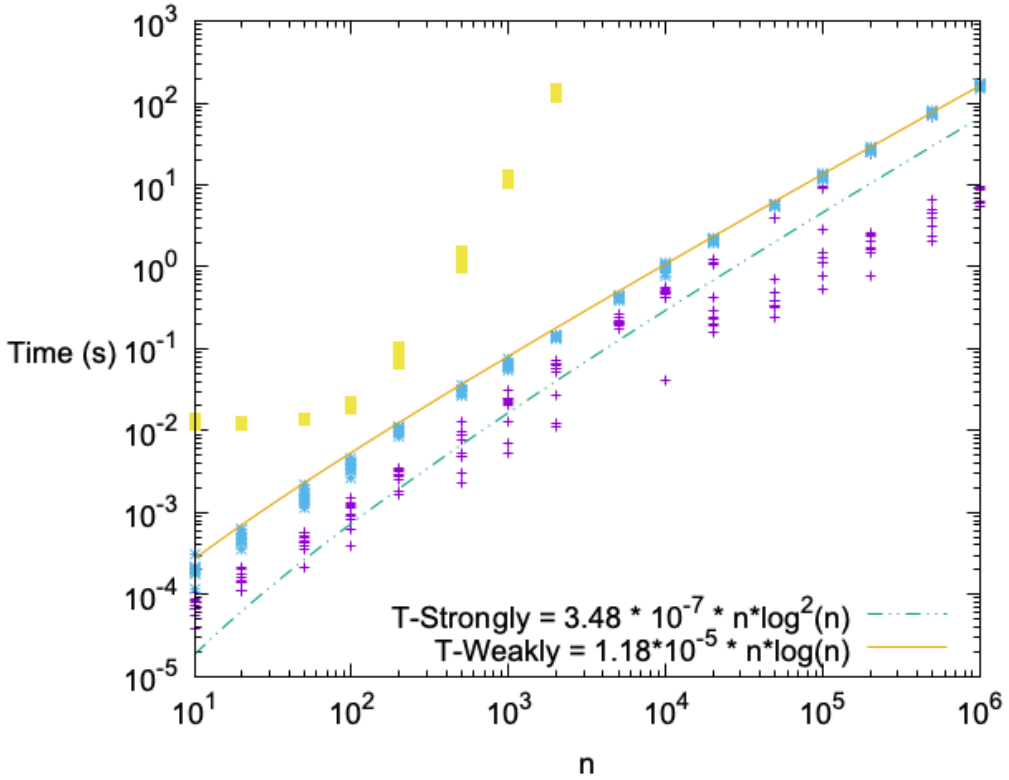


Figure 6.1: Average Time(s) v.s. Input Size(n) of uniformly generated nested constraints. T-Weakly: $\times$. T-Strongly: $+$. T-Mosek: $\square$.

PUC-Rio - Certificação Digital Nº 1621891/CA

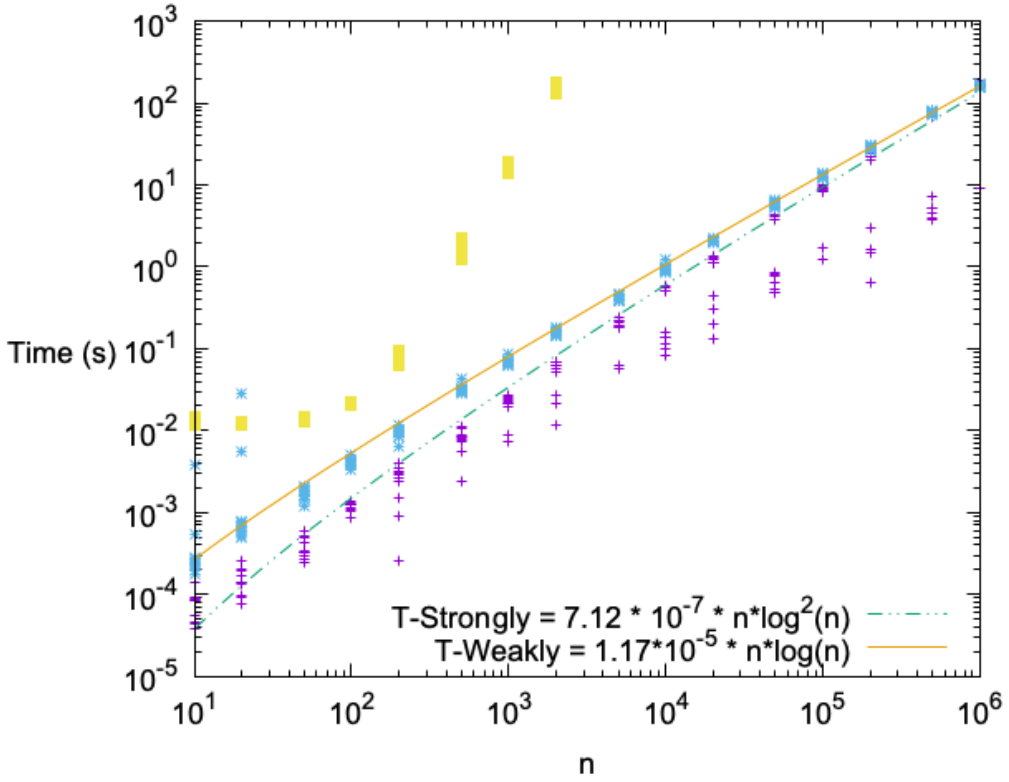


Figure 6.2: Average Time(s) v.s. Input Size(n) of decreasing nested constraints. T-Weakly: $\times$. T-Strongly: $+$. T-Mosek: $\square$.

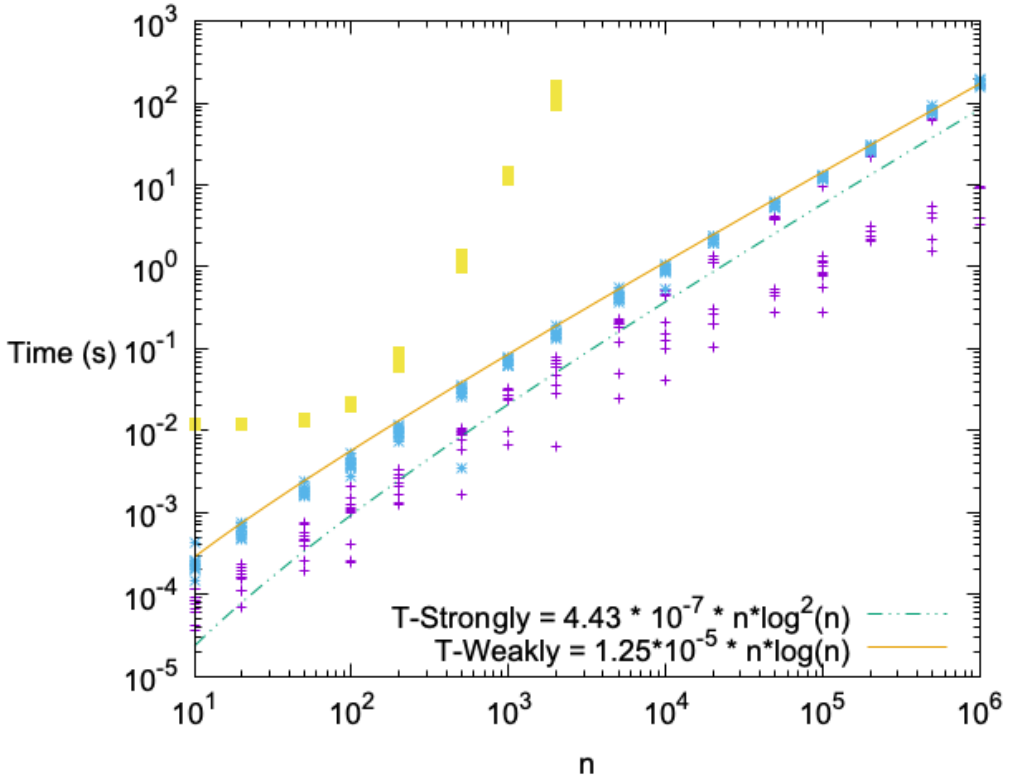


Figure 6.3: Average Time(s) v.s. Input Size(n) of increasing nested constraints. T-Weakly: $\times$. T-Strongly: $+$. T-Mosek: $\square$.

PUC-Rio - Certificação Digital Nº 1621891/CA

	CPU Time(s)		
n	Weakly	Strongly	MOSEK
10	1.41×10^{-4}	7.28×10^{-5}	8.88×10^{-3}
20	4.88×10^{-4}	1.52×10^{-4}	1.20×10^{-2}
50	1.58×10^{-3}	4.39×10^{-4}	1.36×10^{-2}
100	3.69×10^{-3}	1.00×10^{-3}	2.04×10^{-2}
200	9.95×10^{-3}	2.80×10^{-3}	7.86×10^{-2}
500	3.00×10^{-2}	6.87×10^{-3}	1.24
$1,000$	6.31×10^{-2}	1.92×10^{-2}	1.24×10
$2,000$	1.41×10^{-1}	4.86×10^{-2}	1.39×10^{2}
$5,000$	4.25×10^{-1}	2.13×10^{-1}	–
$10,000$	9.71×10^{-1}	4.43×10^{-1}	–
$20,000$	2.11	5.22×10^{-1}	–
$50,000$	5.69	1.10	–
$100,000$	1.26×10	3.70	–
$200,000$	2.65×10	6.34	–
$500,000$	7.54×10	1.72×10	–
$1,000,000$	1.64×10^{2}	7.29×10	–

Table 6.1: Average CPU time for experiments from uniform data

PUC-Rio - Certificação Digital Nº 1621891/CA

	CPU Time(s)		
n	Weakly	Strongly	MOSEK
10	6.24×10^{-4}	7.57×10^{-5}	1.28×10^{-2}
20	3.88×10^{-3}	1.51×10^{-4}	1.21×10^{-2}
50	1.69×10^{-3}	3.83×10^{-4}	1.36×10^{-2}
100	4.12×10^{-3}	1.19×10^{-3}	2.15×10^{-2}
200	9.35×10^{-3}	2.44×10^{-3}	8.05×10^{-2}
500	3.20×10^{-2}	7.88×10^{-3}	1.70
1000	6.96×10^{-2}	2.03×10^{-2}	1.58×10
2000	1.57×10^{-1}	4.96×10^{-2}	1.58×10^{2}
5000	4.18×10^{-1}	1.76×10^{-1}	–
10000	9.49×10^{-1}	2.91×10^{-1}	–
20000	2.08	8.30×10^{-1}	–
50000	5.86	1.69	–
100000	1.28×10	7.51	–
200000	2.86×10	1.45×10	–
500000	7.60×10	3.20×10	–
1000000	1.62×10^{2}	1.50×10^{2}	–

Table 6.2: Average CPU time for experiments from decreasing data

In all of the problem sets, the results suggests that both weakly and strongly polynomial algorithms outperform MOSEK's IP method. Though the results cannot attest to the proposed algorithms accuracy for larger input size ($n > 2000$), both strongly and weakly polynomial algorithms obtained the same exact answers throughout all of the experiments. The strongly polynomial

	CPU Time(s)		
n	Weakly	Strongly	MOSEK
10	2.42×10^{-4}	7.73×10^{-5}	2.58×10^{-4}
20	5.87×10^{-4}	1.58×10^{-4}	1.89×10^{-4}
50	1.85×10^{-3}	4.82×10^{-4}	3.15×10^{-4}
100	3.92×10^{-3}	1.01×10^{-3}	8.92×10^{-4}
200	9.59×10^{-3}	2.35×10^{-3}	8.87×10^{-3}
500	2.82×10^{-2}	8.22×10^{-3}	1.32×10^{-1}
1000	6.91×10^{-2}	2.42×10^{-2}	1.10
2000	1.53×10^{-1}	5.24×10^{-2}	1.96×10
5000	4.35×10^{-1}	1.62×10^{-1}	
10000	9.02×10^{-1}	3.05×10^{-1}	–
20000	2.13	7.05×10^{-1}	–
50000	5.67	2.52	–
100000	1.24×10	1.74	–
200000	2.74×10	1.28×10	
500000	7.92×10	3.81×10	–
1000000	1.74×10^2	8.49×10	–

Table 6.3: Average CPU time for experiments from increasing data

PUC-Rio - Certificação Digital Nº 1621891/CA

algorithm is more efficient than the weakly polynomial. Furthermore, the running time of the strongly polynomial algorithm, is highly volatile for larger input sizes, as shown for $n \geqslant 10,000$, where the standard deviation is at least the same order of magnitude as the average running time. This could be due to quicksort's worst time complexity of $O(n^2)$. Moreover, the strongly polynomial algorithm doesn't always run the sorting of quicksort to completion. If an optimal solution is found before the end of quicksort, the algorithm terminates and returns the solution.

It is worth noting that although the average running times of the strongly polynomial algorithm were superior to that of the weakly polynomial algorithm, there were instances where both algorithms had similar running times. What set the strongly algorithm apart from the weakly algorithm are the instances that were solved at a fraction of the weakly's running time. This is reflected on the standard deviation of the strongly polynomial algorithm running times.

The running times of the weakly polynomial algorithm, can be modelled by a function $T(n) = O(n \log n)$. This is likely due to the way the data was generated. Since the weights and linear constraint terms were generated in a fixed range, $c_i \in [0, 1]$ and $p_i \in [-1, 1]$, it limited the range between the minimum intersection and maximum intersection. Given that we ran the feasibility check to obtain Λ prior to running the weakly polynomial algorithm, it is expected that the factor $O(\log \Lambda)$ remained roughly constant throughout

the experiment. This effectively resulted in a practical $O(n \log n)$ complexity for the weakly polynomial algorithm.

Furthermore, there are no significant differences in running times among the problem sets. The Increasing and Decreasing problem sets, have a similar upper nested constraint structure, i.e. the distance between the upper nested limits are monotonic. This additional structure did not benefit nor hinder the algorithms running time, as all three algorithms had similar running times across all problem sets.

PUC-Rio - Certificação Digital Nº 1621891/CA

7
Conclusion

We have proposed both a weakly and strongly polynomial algorithm with complexity $O(n \log n \log |\Lambda|)$ and $O(n \log n)$ respectively. It is a dual approach, where the dual space is searched for the optimal dual solution. While the weakly polynomial algorithm uses binary search on an interval defined in dual space, the strongly polynomial algorithm utilizes the search procedure developed for the SSP to search over the breakpoints of LR. Our experimental results highlights a significant improvement upon the general IP method.

More research is necessary to adapt the linear time count inversion approximation, developed by (Cole1989), to the weakly polynomial algorithm. If possible, this could reduce the current complexity of $O(n \log n \log |\Lambda|)$ to $O(n \log |\Lambda|)$. An implementation of (Goodrich2013) practical framework of parametric search to our problem could provide a more efficient algorithm in practice, as well as reduce the running time volatility observed in our experiment.

A straightforward extension of the methods developed here could allow to solve a more general class of problems, namely by introducing lower nested constraints and box constraints on the variables. In this regard, Algorithm 3.1.1 should be substituted by a variant of the decomposition algorithm of (Vidal2018). The dual search in both weakly and polynomial algorithms would remain the same. This would result in complexity $O(n \log n \log \frac{|\Lambda|}{|I|})$ for the weakly polynomial algorithm and $O(n \log n \log m)$ for the strongly polynomial algorithm, with $O(n \log^2 n \log m)$ for the implementation used here.

In regards to a more general objective function, it is not clear whether the approach described here using parametric search is adequate for the RAP-LB-L with a quadratic objective function. This is due to the necessity of the technique to be monotonically dependent on the dual parameter λ. In the quadratic case, the breakpoints would also depend on how much resource is allocated to each variable. As for the weakly polynomial approach described, it was based on the assumption that each breakpoint had one and only one intersection. An analogous assumption would be required for the quadratic case, where the multiplicity of the intersections need to be considered.

PUC-Rio - Certificação Digital Nº 1621891/CA

Bibliography

[Agarwal1994] AGARWAL, P.; SHARIR, M. ; TOLEDO, S.. **Applications of parametric searching in geometric optimization**. Journal of Algorithms, 17(3):292 – 318, 1994.

[Agarwal1998] AGARWAL, P. K.; SHARIR, M.. **Efficient algorithms for geometric optimization**. ACM Comput. Surv., 30(4):412–458, Dec. 1998.

[Ajtai1983] AJTAI, M.; KOMLÓS, J. ; SZEMERÉDI, E.. **An $o(n \log n)$ sorting network**. In: PROCEEDINGS OF THE FIFTEENTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC '83, p. 1–9, New York, NY, USA, 1983. ACM.

[Akhil2014] AKHIL, P. T.; SINGH, R. ; SUNDARESAN, R.. **A polymatroid approach to separable convex optimization with linear ascending constraints**. In: 2014 TWENTIETH NATIONAL CONFERENCE ON COMMUNICATIONS (NCC), p. 1–5, Feb 2014.

[Anstreicher1999] ANSTREICHER, K.. **Linear programming in $o((n^3/\log n)l)$ operations**. SIAM Journal on Optimization, 9(4):803–812, 1999.

[Brucker1984] BRUCKER, P.. **An o(n) algorithm for quadratic knapsack problems**. Operations Research Letters, 3(3):163 – 166, 1984.

[Brönnimann1998] BRÖNNIMANN, H.; CHAZELLE, B.. **Optimal slope selection via cuttings**. Computational Geometry, 10(1):23 – 29, 1998.

[Chazelle1992] CHAZELLE, B.; EDELSBRUNNER, H.; GUIBAS, L. ; SHARIR, M.. **Diameter, width, closest line pair, and parametric searching**. In: PROCEEDINGS OF THE EIGHTH ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, SCG '92, p. 120–129, New York, NY, USA, 1992. ACM.

[Chemla2013] CHEMLA, D.; MEUNIER, F. ; CALVO, R. W.. **Bike sharing systems: Solving the static rebalancing problem**. Discrete Optimization, 10(2):120 – 146, 2013.

PUC-Rio - Certificação Digital Nº 1621891/CA

[Coelho2014] COELHO, L. C.; CORDEAU, J.-F. ; LAPORTE, G.. **Thirty years of inventory routing**. Transportation Science, 48(1):1–19, 2014.

[Cole1987] COLE, R.. **Slowing down sorting networks to obtain faster sorting algorithms**. J. ACM, 34(1):200–208, Jan. 1987.

[Cole1989] COLE, R.; SALOWE, J. S.; STEIGER, W. L. ; SZEMERÉDI, E.. **An optimal-time algorithm for slope selection**. SIAM Journal on Computing, 18(4):792–810, 1989.

[D'Amico2014] D'AMICO, A. A.; SANGUINETTI, L. ; PALOMAR, D. P.. **Convex separable problems with linear constraints in signal processing and communications**. IEEE Transactions on Signal Processing, 62(22):6045–6058, Nov 2014.

[Goodrich2013] GOODRICH, M. T.; PSZONA, P.. **Cole's parametric search technique made practical**. CoRR, abs/1306.3000, 2013.

[Hartl2016] HARTL, R. F.; ROMAUCH, M.. **Notes on the single route lateral transhipment problem**. Journal of Global Optimization, 65(1):57–82, May 2016.

[Hochbaum1994] HOCHBAUM, D. S.. **Lower and upper bounds for the allocation problem and other nonlinear optimization problems**. Math. Oper. Res., 19(2):390–409, May 1994.

[Ibaraki1988] IBARAKI, T.; KATOH, N.. **Resource Allocation Problems: Algorithmic Approaches**. MIT Press, Cambridge, MA, USA, 1988.

[Karmarkar1984] KARMARKAR, N.. **A new polynomial-time algorithm for linear programming**. Combinatorica, 4(4):373–395, Dec 1984.

[Katz1993] KATZ, M. J.; SHARIR, M.. **Optimal slope selection via expanders**. Information Processing Letters, 47(3):115 – 122, 1993.

[Lubotzky2012] LUBOTZKY, A.. **Expander graphs in pure and applied mathematics**. Bulletin of the American Mathematical Society, 49(1):113–162, may 2012.

[Megiddo1981] MEGIDDO, N.. **An application of parallel computation to sequential computation: the problem of cost-effective resource allocation**. Technical Report TWISK 202, National Research Institute for Mathematical Sciences, Council for Scientific and Industrial Research (CSIR), Pretoria, South Africa, March 1981.

PUC-Rio - Certificação Digital Nº 1621891/CA

[Megiddo1983] MEGIDDO, N.. **Applying parallel computation algorithms in the design of serial algorithms**. J. ACM, 30(4):852–865, Oct. 1983.

[Megiddo1984] MEGIDDO, N.. **Linear programming in linear time when the dimension is fixed**. J. ACM, 31(1):114–127, Jan. 1984.

[Megiddo1993] MEGIDDO, N.; TAMIR, A.. **Linear time algorithms for some separable quadratic programming problems**. Operations Research Letters, 13(4):203 – 211, 1993.

[Patriksson2008] PATRIKSSON, M.. **A survey on the continuous nonlinear resource allocation problem**. European Journal of Operational Research, 185(1):1 – 46, 2008.

[Patriksson2015] PATRIKSSON, M.; STRÖMBERG, C.. **Algorithms for the continuous nonlinear resource allocation problem?new implementations and numerical studies**. European Journal of Operational Research, 243(3):703 – 722, 2015.

[Ravi1996] RAVI, R.; GOEMANS, M. X.. **The constrained minimum spanning tree problem**. In: Karlsson, R.; Lingas, A., editors, ALGORITHM THEORY — SWAT'96, p. 66–75, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Reischuk1985] REISCHUK, R.. **Probabilistic parallel algorithms for sorting and selection**. SIAM Journal on Computing, 14(2):396–409, 1985.

[Renegar1987] RENEGAR, J.. **On the worst-case arithmetic complexity of approximating zeros of polynomials**. Journal of Complexity, 3(2):90 – 113, 1987.

[Rockafellar1993] ROCKAFELLAR, R.. **Lagrange multipliers and optimality**. SIAM Review, 35(2):183–238, 1993.

[Sen1968] SEN, P. K.. **Estimates of the regression coefficient based on kendall's tau**. Journal of the American Statistical Association, 63(324):1379–1389, 1968.

[Vidal2016] VIDAL, T.; JAILLET, P. ; MACULAN, N.. **A decomposition algorithm for nested resource allocation problems**. SIAM Journal on Optimization, 26(2):1322–1340, 2016.

PUC-Rio - Certificação Digital Nº 1621891/CA

[Vidal2018] VIDAL, T.; GRIBEL, D. ; JAILLET, P.. **Separable convex optimization with nested lower and upper constraints**. INFORMS Journal on Optimization, Articles in Advance, 2018.

[vanOostrum2004] VAN OOSTRUM, R.; VELTKAMP, R. C.. **Parametric search made practical**. Computational Geometry, 28(2):75 – 88, 2004. Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002.

PUC-Rio - Certificação Digital Nº 1621891/CA

A
Proof of $W(\lambda)$ being Convex

The set $W(\lambda)$ is the set of solutions that solve the piecewise-linear concave function g defined in Section 3.3, at a value λ. If λ is not a breakpoint, then $W(\lambda)$ is a single point, and it is trivially convex. As such, we assume that λ is a breakpoint. Then, there exists at least one set of variables K_1 such that their respective weights, $c_i + \lambda p_i$, for $i \in K_1$ are all equal.

Due to λ being a breakpoint, there is an amount of resource Δ_j that must be allocated among the $|K_j|$ variables. Any such allocation does not change the value of the objective function, as they each have equal weights among them. Therefore every point, $\mathbf{x}$ in $W(\lambda)$ must respect the following constraint:

$$\sum_{k \in K_j} x_k = \Delta_j \tag{A-1}$$

This implies that if $\mathbf{x} \in W(\lambda)$, then $\mathbf{x}$ satisfies the set of equations (A-1). It also implies that there are stationary variables in $W(\lambda)$, such that if $i \notin \cup_j K_j$, then x_i has the same value for any $\mathbf{x} \in W(\lambda)$.

Finally, given two points $\mathbf{a}, \mathbf{b} \in W(\lambda)$, then we show that $\mathbf{c} = \mu\mathbf{a} + (1 - \mu)\mathbf{b} \in W(\lambda)$, for $\mu \in (0, 1)$.

We separate the variables into two disjoint sets and treat them separately: The set $\cup_j K_j$ and the set $Q = [n] - \cup_j K_j$. For any stationary $i \in Q$, $a_i = b_i$, for any $i \in \cup_j K_j$, then:

$$\sum_{k \in K_j} \mu a_k + (1 - \mu) b_k \tag{A-2}$$

$$\Leftrightarrow \mu \sum_{k \in K_j} a_k + (1 - \mu) \sum_{k \in K_j} b_k \tag{A-3}$$

$$\Leftrightarrow \mu\Delta_j + (1 - \mu)\Delta_j = \Delta_j \tag{A-4}$$

Given that the same amount of resource, Δ_j is allocated to each set of variables K_j and $\mathbf{c}$ has the same stationary variables as $\mathbf{a}$ and $\mathbf{b}$, we conclude that $\mathbf{c} \in W(\lambda)$, $W(\lambda)$ is convex, as desired.

PUC-Rio - Certificação Digital Nº 1621891/CA

B
Optimal Breakpoint

Here we show that there is at least one breakpoint of g that is optimal. We begin by noting that the definition of subgradients imply that, at a breakpoint, λ, of g, the subgradient is either a point or an interval. This and a stronger assertion is given in the following proposition.

Proposition B.1 *If g is a piecewise-linear concave function, the slope of the two adjacent lines of a breakpoint define the subdifferential at the breakpoint.*

The proof is detailed in Appendix B.1. With the subdifferential's interval clearly determined, we show the sufficient conditions for a breakpoint to be optimal. Namely, if λ_i is the i-th breakpoint of g, then it is optimal if the following inequalities are true.

$$\frac{\mathrm{d}g_L}{\mathrm{d}\lambda}(\lambda_i) \geqslant 0 \tag{B-1}$$

$$\frac{\mathrm{d}g_R}{\mathrm{d}\lambda}(\lambda_i) \leqslant 0 \tag{B-2}$$

$$\tag{B-3}$$

The conditions follow from Proposition B.1 and Property 3.2. From Property 3.2, $0 \in \partial g(\lambda^*)$. From Proposition B.1, $\partial g(\lambda_i) = [\frac{\mathrm{d}g_R}{\mathrm{d}\lambda}(\lambda_i), \frac{\mathrm{d}g_L}{\mathrm{d}\lambda}(\lambda_i)]$. It follows that if λ_i is optimal, then $0 \in \partial g(\lambda_i)$ which is equivalent to inequalities (B-1) and (B-2).

The conditions established in (B-1) and (B-2) do not guarantee the existence of a breakpoint that satisfies them. To show that such a breakpoint always exist, we rely on problem feasibility, and g being concave. From problem feasibility, there exists an optimal λ^* to g. In particular, from concativity there exists λ_L, and λ_R with $\lambda_L < \lambda^*$ and $\lambda_R > \lambda^*$, such that $\frac{\mathrm{d}g}{\mathrm{d}\lambda}(\lambda_L) > 0$ and $\frac{\mathrm{d}g}{\mathrm{d}\lambda}(\lambda_R) < 0$. There are then, only two possible situations that satisfy problem feasibility and concativity. Either one of the linear segmets has gradient zero, or λ^* is a breakpoint. In both situations, inequalities (B-1) and (B-2) are satisfied.

This results in Property 3.3.

PUC-Rio - Certificação Digital Nº 1621891/CA

C
Proof of Proposition B.1

Proof. To see that this is true, let us consider two adjacent line segments of g, g_L and g_R. Let λ_i be the i-th intersection of g and, specifically, the intersection of the two line segments, g_L and g_R. Their equations are:

$$g_L(\lambda) = g_L(\lambda_i) + \frac{\mathrm{d}g_L}{\mathrm{d}\lambda}(\lambda_i)(\lambda - \lambda_i) \tag{C-1}$$

$$g_R(\lambda) = g_R(\lambda_i) + \frac{\mathrm{d}g_R}{\mathrm{d}\lambda}(\lambda_i)(\lambda - \lambda_i) \tag{C-2}$$

Where g_L is defined in $[\lambda_{i-1}, \lambda_i]$, for $\lambda_{i-1} < \lambda_i$ and g_R is defined in $[\lambda_i, \lambda_{i+1}]$ for some $\lambda_{i+1} > \lambda_i$. Note that the derivatives of both g_L and g_R are constant because they're lines. Clearly, the derivatives of g_L and g_R are subgradients at λ_i. Moreover, by concativity, $\frac{\mathrm{d}g_L}{\mathrm{d}\lambda} > \frac{\mathrm{d}g_R}{\mathrm{d}\lambda}$.

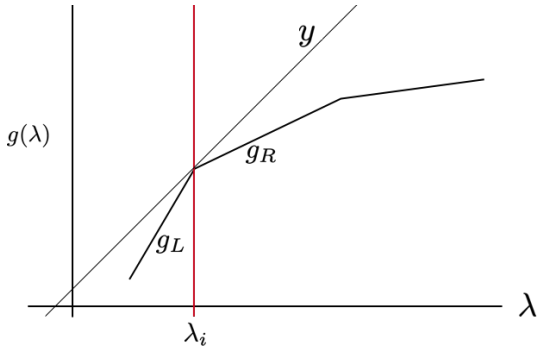


Figure C.1: Function g, it's line segments g_L and g_R, and line y.

Let $v \in [\frac{\mathrm{d}g_R}{\mathrm{d}\lambda}, \frac{\mathrm{d}g_L}{\mathrm{d}\lambda}]$, and let y be the equation of the line with gradient v that intersects point $(\lambda_i, g(\lambda_i))$. If $\lambda < \lambda_i$:

PUC-Rio - Certificação Digital Nº 1621891/CA

$$v \leqslant \frac{\mathrm{d}g_L}{\mathrm{d}\lambda}(\lambda_i) \tag{C-3}$$

$$\Leftrightarrow \langle v, (\lambda - \lambda_i)\rangle \geqslant \langle \frac{\mathrm{d}g_L}{\mathrm{d}\lambda}(\lambda_i), (\lambda - \lambda_i)\rangle \tag{C-4}$$

$$\Leftrightarrow g_L(\lambda_i) + \langle v, (\lambda - \lambda_i)\rangle \geqslant g_L(\lambda_i)\langle \frac{\mathrm{d}g_L}{\mathrm{d}\lambda}(\lambda_i), (\lambda - \lambda_i)\rangle \tag{C-5}$$

$$\Leftrightarrow y(\lambda) \geqslant g_L(\lambda) \tag{C-6}$$

$$\tag{C-7}$$

Thus, the line y is always situated above line g_L for $\lambda < \lambda_i$. On the other hand, if $\lambda > \lambda_i$:

$$v \geqslant \frac{\mathrm{d}g_R}{\mathrm{d}\lambda}(\lambda_i) \tag{C-8}$$

$$\Leftrightarrow \langle v, (\lambda - \lambda_i)\rangle \geqslant \langle \frac{\mathrm{d}g_R}{\mathrm{d}\lambda}(\lambda_i), (\lambda - \lambda_i)\rangle \tag{C-9}$$

$$\Leftrightarrow g_R(\lambda_i) + \langle v, (\lambda - \lambda_i)\rangle \geqslant g_R(\lambda_i)\langle \frac{\mathrm{d}g_R}{\mathrm{d}\lambda}(\lambda_i), (\lambda - \lambda_i)\rangle \tag{C-10}$$

$$\Leftrightarrow y(\lambda) \geqslant g_R(\lambda) \tag{C-11}$$

We conclude that the line y is always situated above g, except at λ_i, where they intersect. Thus, $v \in [\frac{\mathrm{d}g_R}{\lambda}, \frac{\mathrm{d}g_L}{\lambda}]$, is indeed a subgradient at $g(\lambda_i)$. ■

PUC-Rio - Certificação Digital Nº 1621891/CA