Efficient Resource Allocation with Non-Concave Objective Functions

Arne Andersson
Computing Science Department
Information Technology
Uppsala University
Box 311
SE - 751 05 Uppsala, Sweden
arnea@csd.uu.se
http://www.csd.uu.se/˜arnea

Fredrik Ygge
EnerSearch AB
Chalmers Science Park
SE - 412 88 Gothenburg, Sweden
Fredrik.ygge@enersearch.se
http://www.enersearch.se/ygge

Submission for Computational Optimization and Applications

Abstract

We consider resource allocation with separable objective functions defined over subranges of the integers. While it is well known that (the maximization version of) this problem can be solved efficiently if the objective functions are concave, the general problem of resource allocation with non-concave functions is difficult. In this article we show that for fairly well-shaped non-concave objective functions, the optimal solution can be computed efficiently. Our main enabling ingredient is an algorithm for aggregating two objective functions, where the cost depends on the complexity of the two involved functions. As a measure of complexity of a function, we use the number of subintervals that are convex or concave.

1 Introduction

1.1 The Problem

We consider resource allocation with separable objective functions defined over subranges of the integers.

An important tool is the aggregation of two objective functions into one. Given two functions f_1, f_2 defined over subintervals of length I_1 and I_2, compute the aggregated function f_t, defined over a subinterval of length (at most)[1] $I_1 + I_2$:

$$f_t(r) = \max_{r_1+r_2=r} f_1(r_1) + f_2(r_2), \tag{1}$$

and for each r in the subinterval compute the corresponding r_1 and r_2.

Pair-wise aggregation is a useful method also for problems with a large number of objective functions, in particular when these functions are non-concave. A typical algorithm is to compute the global objective function by incorporating the functions one by one [Ibaraki and Katoh, 1988]. (Another approach is to aggregate the functions pair-wise in a binary tree fashion; this method is particularly useful in distributed and dynamic environments [Andersson and Ygge, 1998].)

The complexity of aggregating two objective functions depends on the properties of the functions. Two cases are distinguished and thoroughly studied in the literature:

[1] If the total available resource to be allocated is smaller than $I_1 + I_2$, then the aggregated function need not be computed over the entire interval $I_1 + I_2$.

- The general case, when no assumptions can be made about the functions. Then an aggregation requires $\Theta(I_1 I_2)$ time.
- The special case of two concave functions. Then the aggregation can be made much faster, in $\Theta(I_1+I_2)$ time.

As the domain of the objective functions may be very large compared to the number of functions, the difference between these two complexities will be significant in many applications.

Although it may be hard to find realistic examples where arbitrary objective functions appear, it is rather easy to come up with examples of non-concave objective functions. As one example, we may consider objective functions that are zero up to some limit value and then concave. As another example, taken from an industrial application, we may consider a power plant with two generators, with peaks at half and full capacity but with a lower value inbetween (it may simply be very costly or even impossible to run a generator at half its capacity).

Although such objective functions are fairly well-shaped, traditional resource allocation algorithms would not take advantage of this and the cost of aggregation would be high.

1.2 Main result

In this article, we present a new algorithm for aggregating objective functions. The main point is that the complexity of the aggregation is adaptive to the shape of the two functions to be aggregated; the simpler curves the lower complexity. The main theorem is the following:

Theorem 1 *Let f_1 and f_2 be two objective functions defined over intervals of length I_1 and I_2. Furthermore, assume that the two functions can be divided into s_1 and s_2 segments respectively, such that each segment is either convex or concave. Then, the aggregated objective function and the corresponding optimal allocations can be computed in $O(I_1 s_2 + I_2 s_1)$ time.*

The bound is obtained by dividing the two functions into (convex or concave) segments and finding all candidates (strictly defined in Section 2) for the optimal allocation for each amount of resource in all pairs of segments. The main technical contribution is that all candidates can be efficiently found not only in two concave segments, but also in two convex as well as in one convex and one concave segment. (The search for all candidates in two convex segments is straightforward while the search in one convex and one concave segment is non-trivial.)

The complexity in Theorem 1 has two major applications. First, it improves the complexity of finding the optimal allocation for non-concave objective functions. Secondly, it allows for more accurate approximations of arbitrary objective functions. For example, if the two objective functions are approximated by polynomes with degree p, their approximations can be aggregated in $O(p(I_1+I_2))$ time. Hence, instead of approximating each objective function by one concave segment, more accurate approximations can be used.

1.3 A note on the complexity of the general problem

It is worth noting that in most formulations of the resource allocation problem, the goal is not to compute a global objective function over the entire domain, but to compute the optimal allocation (and maybe the total value of this allocation) for a specific total resource. In that case, it may seem like a waste of time to compute the entire function. Indeed, if the objective functions are concave, a binary-search type algorithm solves the problem more efficient. The complexity is $O(n \log(I/n))$, where each objective is assumed to be defined over an interval of length I, and n is the number of objective functions. (In some cases the complexity can be reduced further by the use of, for example, a Newton-Raphson algorithm [Press *et al.*, 1994].)

However, for the general non-concave case, the situation is worse. In the textbook by Ibaraki and Katoh [Ibaraki and Katoh, 1988] it is pointed out that the general problem with *non-separable* objective functions is NP-hard. It is also stated that it is unknown whether the problem with separable functions is NP-hard or not. However, there is a simple proof that this problem is also NP-hard. As we have not found any proof in the literature, we include a sketch here.

A reduction can be made from the knapsack problem: Given n weights $w_1, \ldots, w_n$ and a target W, determine if the target can be expressed as a sum of weights. This problem can be reduced to resource allocation with separable objective functions: let $f_i(w_i) = w_i$ and let f_i be 0 for all other values. Let W be the total resource and solve the resource allocation problem. If and only if the total value is W, the knapsack problem has a solution. Hence, resource allocation with separable objective functions is NP-hard.

Therefore, we can not hope for any exceptionally good algorithm for the general case, even when the functions are separable.

2 The Aggregation Algorithm

Recall the definitions of f_1, f_2, f_t, I_1 and I_2 from Eq. (1). The domain intervals of the two functions f_1 and f_2 to be aggregated are $[R_1^{min} \ldots R_1^{max}]$ and $[R_2^{min} \ldots R_2^{max}]$ respectively, i.e. $I_i = R_i^{max} - R_i^{min} + 1$, $i = 1, 2$. Let f_t be represented by a vector $\mathbf{f}_t = [f_t^1, f_t^2, \ldots, f_t^{I_1+I_2}]$, and let the optimal allocations of r_1 and r_2 for each r be represented by the vectors $\mathbf{r}_1 = [r_1^1, r_1^2, \ldots, r_1^{I_1+I_2}]$and $\mathbf{r}_2 = [r_2^1, r_2^2, \ldots, r_2^{I_1+I_2}]$ respectively.[2] As an example, f_t^7 corresponds to the optimal utility for $r = R_1^{min} + R_2^{min} + 6$, and r_1^7 and r_2^7 represent the optimal allocation at this r. The solution to an example problem is shown in Figure 1. Each filled circle in the figure is an optimal allocation for a certain total resource, i.e. is an element in the $\mathbf{f}_t$ vector.

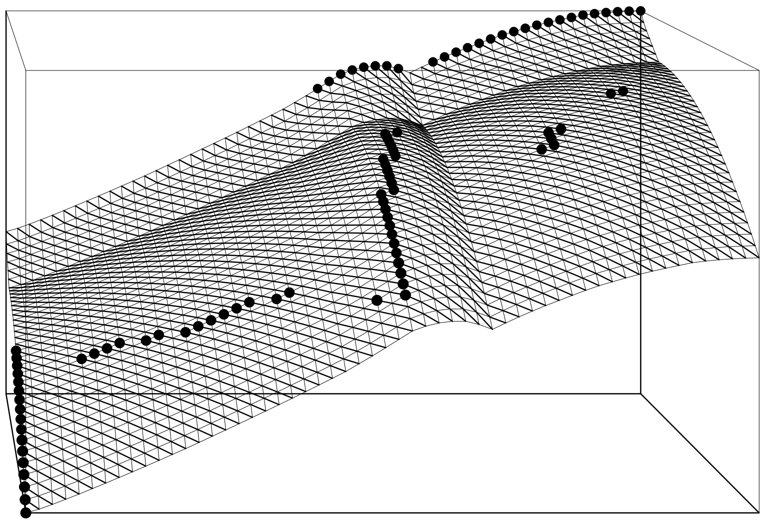

Figure 1: *The aggregation of two functions. The aggregated function is constituted by the optimal allocations (illustrated as filled circles) of the respective amounts of total resource.*

We use Algorithm 1 to compute $\mathbf{f}_t$, $\mathbf{r}_1$, and $\mathbf{r}_1$.

The division of the functions in Figure 1 into concave and convex segments (one important step in Algorithm 1) is shown in Figure 2. The figure also illustrates local candidates (defined below). As is seen from Figure 1 and Figure 2 it is – for this example – sufficient to try all local candidates in order to find the globally optimal allocations. Below we will prove this for the general case.

We use Δ to denote function differences.

[2] Again if the total resource which can be allocated is smaller than $I_1 + I_2$ smaller vectors can be used.

For every element, i, of the $\mathbf{f}_t$ vector
  Set $f_t^i \leftarrow \min(f_1) + \min(f_2)$ //or some other sufficiently small number
Divide f_1 and f_2 into concave and convex segment
For every combination of segments with one segment from f_1 and one from f_2
  For every local candidate allocation (defined below), (r_1, r_2), of the two segments
    If $f_t^{r_1+r_2} < f_1(r_1) + f_2(r_2)$
    {
      $f_t^{r_1+r_2} = f_1(r_1) + f_2(r_2)$
      $r_1^{r_1+r_2} = r_1$
      $r_2^{r_1+r_2} = r_2$
    }

Algorithm 1: ALGORITHM FOR AGGREGATION OF TWO FUNCTIONS.

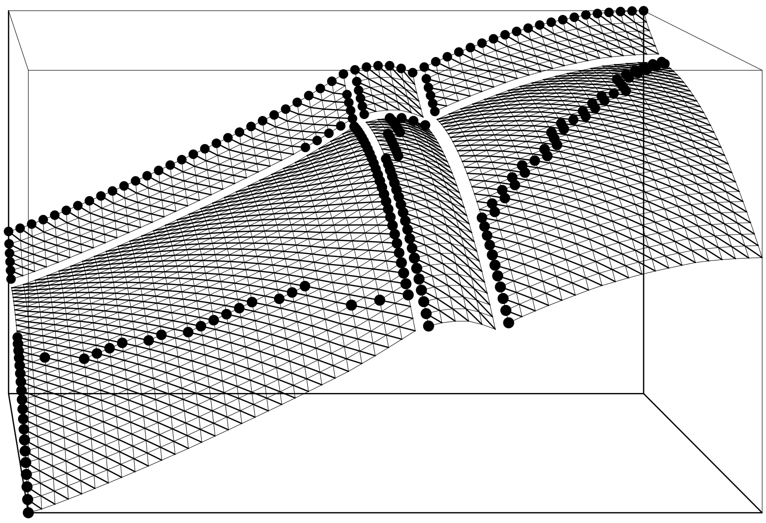

Figure 2: *The functions shown in Figure 1 divided into segments. The functions consists of three and two segments respectively, {convex, concave, concave} and {concave, convex}. For each combination of segments, the local candidates (cf. Definition 3) are shown as filled circles.*

Definition. 1 $\Delta f(r) = f(r+1) - f(r)$

We define a global candidate as satisfying the following criterion (cf. the Kuhn-Tucker criterion [Ibaraki and Katoh, 1988]):

Definition. 2 *An allocation (r_1, r_2) is a global candidate if and only if*

$$r_1 = R_1^{min} \text{ or } r_2 = R_2^{max} \text{ or } \Delta f_1(r_1 - 1) \geq \Delta f_2(r_2) \quad (2)$$

and

$$r_1 = R_1^{max} \text{ or } r_2 = R_2^{min} \text{ or } \Delta f_1(r_1) < \Delta f_2(r_2 - 1). \quad (3)$$

The essential point of the definitions is that it is sufficient to examine the global candidates to perform the aggregation. As a matter of fact, due to the strict inequality in Eq. (3), there are optimal allocations which are not candidates. However, as shown by Lemma 1 these allocations need not be considered.

Lemma 1 *For every optimal allocation of resource r there is a global candidate for resource r with the same total utility.*

Proof. Assume than an allocation, (r_1, r_2), is not a global candidate, then by definition one of the following cases must occur:

1. $r_1 > R_1^{min}$ and $r_2 < R_2^{max}$ and $\Delta f_1(r_1 - 1) < \Delta f_2(r_2)$,
2. $r_1 < R_1^{max}$ and $r_2 > R_2^{min}$ and $\Delta f_1(r_1) > \Delta f_2(r_2 - 1)$, or
3. $r_1 < R_1^{max}$ and $r_2 > R_2^{min}$ and $\Delta f_1(r_1) = \Delta f_2(r_2 - 1)$.

In case 1, $(r_1 - 1, r_2 + 1)$ has higher total utility than (r_1, r_2) and thus (r_1, r_2) is not an optimal allocation. In case 2, $(r_1 + 1, r_2 - 1)$ has higher total utility.

In case 3, we can move one unit of resource from r_2 to r_1, i.e. $(r_1 + 1, r_2 - 1)$, with maintained total utility, i.e. $f_1(r_1) + f_2(r_2) = f_1(r_1 + 1) + f_2(r_2 - 1)$. Now either case 1, 2, or 3 above occur or the allocation is a candidate. If case 1 or 2 occurs the allocation is not optimal. If the new allocation is a candidate there was obviously a candidate with the same utility. If case 3 occurs we can apply the above reasoning recursively. (Termination of the recursion process is obvious.) □

A segment in f_1 is ranging from r_1^{min} (inclusive) to r_1^{max} (inclusive) and correspondingly for a segment in f_2. We define a *local candidate* as follows.

Definition. 3 *An allocation (r_1, r_2) is a local candidate if and only if*

$$r_1 = r_1^{min} \text{ or } r_2 = r_2^{max} \text{ or } \Delta f_1(r_1 - 1) \geq \Delta f_2(r_2) \tag{4}$$

and

$$r_1 = r_1^{max} \text{ or } r_2 = r_2^{min} \text{ or } \Delta f_1(r_1) < \Delta f_2(r_2 - 1). \tag{5}$$

Lemma 2 *Every global candidate is a local candidate for some combination of segments.*

Proof. The proof is rather trivial and we only prove for (r_1, r_2), $r_1 = R_1^{min}$, $\Delta f_1(r_1) < \Delta f_2(r_1 - 1)$ as all other cases are analogous. Obviously, some segment will contain r_1 and it must be the case that $r_1 = r_1^{min}$. Next, if $r_2 = r_2^{min}$ then (r_1, r_2) is a local candidate, else it holds that $\Delta f_1(r_1) < \Delta f_2(r_1 - 1)$, in which case it is also a local candidate. □

Lemma 3 *Algorithm 1 correctly computes f_t and the optimal allocations.*

Proof. The algorithm tests every local candidate for every possible combination of segments. Hence, the lemma follows directly from Lemmas 1 and 2. □

Remaining before having a complete algorithm is to show how to find all local candidates in all possible combination of segments.

3 Finding all Candidates in Convex and Concave Segments

There are three possible combinations of segments: two concave, two convex, and one concave and one convex. In this section we show that all local candidates can be found in linear time for all the three possible combinations. How to find all candidates in two concave functions is well known (or rather how to find the optimal allocation when the objective functions are concave is well known, but we can utilize the same algorithms), and the proof in Section 3.1 is included for the sake of completeness. Finding all candidates in two convex segments turns out to be very simple, whereas finding all candidates in one concave and one convex segment is non-trivial. For convenience we will refer to a local candidate only as a candidate in this section.

3.1 Two Concave Segments

Lemma 4 *If $f_1(r_1)$, $r_1 \in [r_1^{min}..r_1^{max}$ and $f_2(r_2)$, $r_2 \in [r_2^{min}..r_2^{max}$ are concave and (r_1, r_2) is a candidate, then any allocation $(r_1', r_2'), r_1' + r_2' > r_1 + r_2$ for which $r_1^{min} \leq r_1' < r_1, r_2^{min} \leq r_2' \leq r_2^{max}$ or $r_1^{min} \leq r_1' \leq r_1^{max}, r_2^{min} \leq r_2' < r_2$ is not a candidate.*

Proof. It is obvious that $r_i' \geq r_i$ if $r_i = r_i^{min}$, and $r_i' \geq r_i$ if $r_j = r_j^{max}, i \neq j$. Then we need to show that the lemma holds for $r_1^{min} < r_1 \leq r_1^{max}$ and $r_2^{min} \leq r_2 < r_2^{max}$ (analogous reasoning for $r_1^{min} \leq r_1 < r_1^{max}, r_2^{min} < r_2 \leq r_2^{max}$). Assume that $r_1' < r_1$ (analogous for $r_2' < r_2$). It suffices to show that this implies $\Delta f_1(r_1') \geq \Delta f_2(r_2' - 1)$. We have

$$\Delta f_1(r_1') \geq \Delta f_1(r_1 - 1) \geq \Delta f_2(r_2) \geq \Delta f_2(r_2' - 1).$$

The first inequality follows from concavity, the next follows from that (r_1, r_2) is a candidate (and $r_1 \neq r_1^{min}, r_2 \neq r_2^{max}$, and the third from concavity and $r_2' > r_2$ (since $r_1' + r_2' > r_1 + r_2$). □

Hence, Lemma 4 shows that candidates only can be found along the borders of the area covered by the segments and along the type of diagonal shown in the bottom-right combination of Figure 2.

Lemma 5 *All candidates in two concave segments can be found in linear time.*

Proof. We give a constructive proof by giving the following algorithm (cf. algorithm INCREMENT [Ibaraki and Katoh, 1988, pp. 54 - 55]) which finds all candidates:

1. Start with $r_1 = r_1^{min}$ and $r_2 = r_2^{min}$ which is a candidate.
2. If $r_1 < r_1^{max}$ and $\Delta f_1(r_1) \geq \Delta f_2(r_2)$ then increase r_1 by one unit, otherwise increase r_2 by one unit.
3. Repeat 2 until $r_1 = r_1^{max}$ and $r_2 = r_2^{max}$.

From Definition 3 and the fact that the segments are concave we get that every new allocation obtained by Step 2 is a candidate. Lemma 4 implies that all candidates are found by this algorithm. □

3.2 Two Convex Segments

Lemma 6 *All candidates in two convex segments can be found in linear time.*

Proof. All candidates for a given r can be computed in constant time. This is because there are only two possibilities for a candidate:

1. $r_1 = \begin{cases} r - r_2^{min}, & r \le r_1^{max} + r_2^{max} \\ r_1^{max}, & r > r_1^{max} + r_2^{max} \end{cases}$ and $r_2 = r - r_1$, and

2. vice versa, i.e. $r_2 = \begin{cases} r - r_1^{min}, & r \le r_1^{max} + r_2^{max} \\ r_2^{max}, & r > r_2^{max} + r_1^{max} \end{cases}$ and $r_1 = r - r_2$.

This follows from the fact that if none of the above is fulfilled we have that for two convex segments, either $\Delta f_1(r_1) > \Delta f_2(r_2 - 1)$ or $\Delta f_2(r_2) > \Delta f_1(r_1 - 1)$ or $(\Delta f_1(r_1) = \Delta f_2(r_2 - 1)$ and $\Delta f_2(r_2) = \Delta f_1(r_1 - 1))$ holds. In none of these cases the allocation is a candidate.

Then the following algorithm finds all candidates in two convex segments in linear time: For each r, return the two possibilities above. □

Hence, the proof shows that candidates can only be found along the borders of the area covered by combination of the segments, cf. the upper-left combination in Figure 2.

3.3 Concave and Convex Segments

For readability we will relate our description of the algorithm for finding all candidates in one convex and one concave segment to a specific example. The example is given in Figure 3.

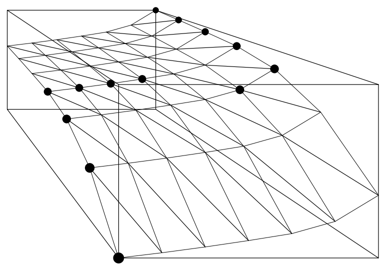

Figure 3: *Finding all candidates in one convex and one concave segment.* $f_1(r_1)$ *is convex for* $r_1 \in [0..6]$ *with* $f_1(r_1) = 0, 9, 19, 30, 42, 63,$ *and* 108 *for the respective values of* r_1. $f_2(r_2)$ *is concave for* $r_2 \in [0..6]$ *with* $f_2(r_2) = 0, 100, 150, 170, 178, 185,$ *and* 191. *All candidates are shown as filled circles.*

An efficient search for all candidates in one convex and one concave segment is based on a number of criteria for excluding possible candidates; these criteria are given in Lemmas 7, 8, and 9. By applying these criteria properly, we can ensure that only a linear number of points need to be considered. This is done in the proof of Lemma 10. (For the reader who feels that there is a tendency to get lost in details, it probably helps to occasionally consult the example in the end of this section (and Figure 7 in particular) in order to understand the roles of the different lemmas and the principles of the algorithm.)

Let r_1 be taken from the convex segment and r_2 from the concave segment.

Lemma 7 *If* $r_1^{min} \le r_1 < r_1^{max}$, $r_2^{min} < r_2 \le r_2^{max}$, $\Delta f_1(r_1) \ge \Delta f_2(r_2 - 1)$, *there is no candidate* (r_1', r_2'), $r_1 \le r_1' < r_1^{max}$, $r_2 \le r_2' \le r_2^{max}$.

Proof. By convexity, $\Delta f_1(r_1') \ge \Delta f_1(r_1)$, and by concavity, $\Delta f_2(r_2' - 1) \ge \Delta f_2(r_2 - 1)$. Hence, $\Delta f_1(r_1') \ge \Delta f_2(r_2' - 1)$. Together with the preconditions $r_1 < r_1^{max}$ and $r_2 > r_2^{min}$, this gives the lemma. □

The lemma is visualized in Figure 4.

Lemma 8 *If* $(r_1, r_2), r_1^{min} < r_1 < r_1^{max}, r_2^{min} < r_2 \le r_2^{max}$ *is a candidate then* (r_1', r_2'), $r_1^{min} < r_1' \le r_1$, $r_2^{min} < r_2' < r_2$, *is not.*

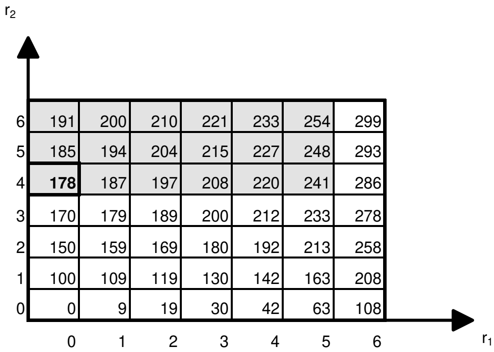

Figure 4: *Illustration of Lemma 7. The number in each cell represents* $f_1(r_1) + f_2(r_2)$ *for the example of Figure 3. At allocation* $(0, 4)$*, it holds that* $\Delta f_1(r_1) \geq \Delta f_2(r_2 - 1)$ *(in numbers* $187 - 178 > 178 - 170$*), and hence no allocation in the shaded area can be a candidate.*

Proof. To prove the lemma, it suffices to show that $\Delta f_1(r'_1 - 1) < \Delta f_2(r'_2)$. We have

$$\Delta f_1(r'_1 - 1) \leq \Delta f_1(r_1 - 1) \leq \Delta f_1(r_1) < \Delta f_2(r_2 - 1) \leq \Delta f_2(r'_2).$$

The first two inequalities follows from convexity, the third from the fact that (r_1, r_2) is a candidate, and the forth from concavity. □

This is illustrated in Figure 5.

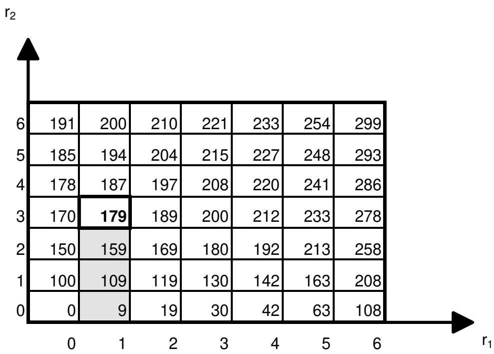

Figure 5: *Illustration of Lemma 8. The allocation* $(1, 3)$ *is a candidate, and hence no allocation in the shaded area can be a candidate.*

Lemma 9 *If* $r_1^{min} < r_1 \leq r_1^{max}$*,* $r_2^{min} \leq r_2 < r_2^{max}$*,* $\Delta f_1(r_1 - 1) < \Delta f_2(r_2)$*, there is no candidate* (r'_1, r'_2)*,* $r_1^{min} < r'_1 \leq r_1$*,* $r_2^{min} \leq r'_2 \leq r_2$*.*

Proof. By convexity, $\Delta f_1(r'_1 - 1) \leq \Delta f_1(r_1 - 1)$, and by concavity $\Delta f_2(r'_2) \geq \Delta f_2(r_2)$. Hence, $\Delta f_1(r'_1 - 1) < \Delta f_2(r'_2)$. Together with the preconditions $r_1 > r_1^{min}$ and $r_2 < r_2^{max}$ this gives the lemma. □

The above lemma is illustrated in Figure 6.

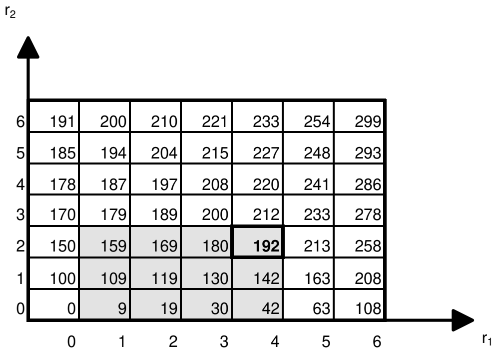

Figure 6: *Illustration of Lemma 9. At allocation* $(4,2)$*, it holds that* $\Delta f_1(r_1-1) < \Delta f_2(r_2)$ *(in numbers* $192-180 < 212-192$*), and hence no allocation in the shaded area can be a candidate.*

Lemma 10 *All candidates in one convex and one concave segment can be found in linear time.*

Proof. We give a constructive proof:

1. (r_1^{min}, r_2^{min}) is a candidate. Set $r_1 \leftarrow r_1^{min}$, $r_2 \leftarrow r_2^{min}$.

2. Repeat $r_2 \leftarrow r_2+1$ until $\Delta f_1(r_1^{min}) \geq \Delta f_2(r_2-1)$ or $r_2 = r_2^{max}+1$.

3. We can now conclude, either because $r_2 = r_2^{max}+1$ or by Lemma 7, that there is no candidate (r_1', r_2'), $r_1^{min} \leq r_1' < r_1^{max}$, $r_2 \leq r_2' \leq r_2^{max}$. In other words: each remaining candidate (r_1', r_2') satisfies either
$$r_1^{min} < r_1' < r_1^{max}, \quad r_2^{min} \leq r_2' < r_2 \tag{6}$$
or
$$r_1 = r_1^{max}. \tag{7}$$

4. In this phase, we consider the remaining candidates fulfilling Eq. (6). Set $r_1 = r_1^{min}+1$ and $r_2 = r_2-1$. We will now maintain the following invariant: Each remaining (i.e. non-visited) candidate (r_1', r_2') must satisfy
$$r_1 \leq r_1' < r_1^{max}, \quad r_2^{min} \leq r_2' \leq r_2$$
Initially, the invariant holds.

   While $r_1 < r_1^{max}$ and $r_2 \geq r_2^{min}$, perform one of the four cases:

   (i) $r_2 = r_2^{min}$, then we can increase r_1 by one; the invariant will still hold,

   (ii) $\Delta f_1(r_1) \geq \Delta f_2(r_2-1)$. Then, by Lemma 7, we can decrease r_2 by one; the invariant will still hold.

   (iii) $r_2 < r_2^{max}$, $\Delta f_1(r_1-1) < \Delta f_2(r_2)$. Then, according to Lemma 9 we can increase r_1 by one; the invariant will still hold.

   (iv) If none of the cases above occur, (r_1, r_2) is a candidate (see below). Then, from Lemma 8, we have that we can increase r_1 by one; the invariant will still hold.

5. When leaving the loop above, the remaining points to investigate are all fulfilling Eq. (7). Start at (r_1^{max}, r_2^{max}) which is a candidate and decrease r_2 (implying that $\Delta f_2(r_2)$ increases) until $\Delta f_1(r_1^{max}-1) < \Delta f_2(r_2)$ or $r_2 = r_2^{min}$. Now, we have from Lemma 9 that no further candidates exist.

The only thing remaining is to prove that during the loop in Step 4 only the four listed cases can occur. For the algorithm to enter the loop, we have that $r_1^{min} < r_1 < r_1^{max}, r_2^{min} \leq r_2 \leq r_2^{max}$. If case (i) does not occur it holds that $r_1^{min} < r_1 < r_1^{max}, r_2^{min} < r_2 \leq r_2^{max}$. Note that for this space $\Delta f_1(r_1)$ and $\Delta f_2(r_2-1)$ are defined and the applicability of Lemma 7 can be tested. If case (ii) does not occur we have that $r_1^{min} < r_1 < r_1^{max}, r_2^{min} < r_2 \leq r_2^{max}, \Delta f_1(r_1) < \Delta f_2(r_2-1)$. If case (iii) does not occur we have either $r_1^{min} < r_1 < r_1^{max}, r_2 = r_2^{max}, \Delta f_1(r_1) < \Delta f_2(r_2-1)$ or $r_1^{min} < r_1 < r_1^{max}, r_2^{min} < r_2 < r_2^{max}, \Delta f_1(r_1) < \Delta f_2(r_2-1), \Delta f_1(r_1-1) \geq \Delta f_2(r_2)$. In both cases (r_1, r_2) is a candidate, and hence case (iv) must occur. □

As the above algorithm finds all candidates, we see that candidates can only exist along the borders and along the type of diagonal shown in the bottom-left diagonal of Figure 2.

An Example In Figure 7 we show how the algorithm works, step by step, for the example of Figure 3.

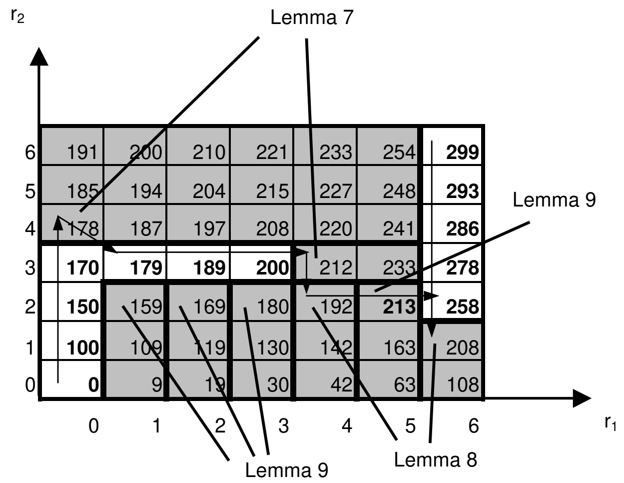

Figure 7: *Illustration of the algorithm for finding all candidates in one convex and one concave segment. The number in each cell represents $f_1(r_1) + f_2(r_2)$ for the example of Figure 3. The path computed by the algorithm is shown by the arrows. The optimal candidates are the non-filled cells (cf. the filled circles in Figure 3), and filled cells denote allocations that can be excluded with support from different lemmas. (***Fredrik comments***: Once section numbering is firm, check the text in the Figure.)*

We start at Step 1, i.e. $(0, 0)$. Then we increase r_2 until Lemma 7 can be applied as described in Step 2. This occurs at $r_2 = 4$. Now at Step 3 we can conclude that all remaining possible candidates are $(r_1', r_2'), 0 < r_1' < 6, 0 \leq r_2' < 4$ and $(r_1', r_2'), r_1' = 6, 0 \leq r_2' \leq 6$, cf. Figure 4. We start with the former space and loop as described by Step 4. Then the first visited allocation is $(1, 3)$. This is a candidate, and hence all allocations below the current allocation in the figure can be excluded, cf. Figure 5. We therefore increase r_1. This is repeated for allocations $(2, 3)$ and $(3, 3)$. At $(4, 3)$ we can apply Lemma 7, and the current allocation and next allocation to the right can be excluded. Hence, we decrease r_2. At this allocation, $(4, 2)$ we can apply Lemma 9, and the current allocation and the two below can be excluded, cf. Figure 6. Therefore, we increase r_1. Then again we have found a candidate and increase r_1. Now $r_1 = 6$. Then we start from the top right corner, Step 5. Now we decrease r_2 until Lemma 9 is applicable, and we are done.

4 Proof of Theorem 1

We use Algorithm 1. The correctness of Algorithm 1 was proven by Lemma 3. As the process of segmentation consists only of comparing the previous slope with the current, it can be performed in linear time. From Lemma 5, Lemma 6, and Lemma 10 we have that for every possible combination, all candidates can be found in linear time. Hence, the theorem follows.

5 Numerical Stability

Problems caused by floating-point arithmetic should always be considered when implementing algorithms of this kind. Fortunately, our algorithms can be implemented to ensure numerical stability. We note that all decisions are made either by considering indices (which are integers) or by comparing Δ-values. Indeed, the outcome of our algorithms is determined only by the relative order of the Δ-values.

Therefore, if we ensure that the same numerical representation of the Δ-values is used throughout the computations, i.e. both when dividing an interval into concave and convex segments and when finding the candidates, the output will be consistent with the Δ-values. A simple way to ensure this is to start the aggregation of two functions by computing the Δ-values and store them in a table.

We may even state the following:

Theorem 2 *If the function f_1 is specified by a table containing $\Delta f_1(r_1^{min}), \ldots, \Delta f_1(r_1^{max} - 1)$ and the function f_2 is specified similarly, then a set of points, containing all candidates, can be found at the cost stated in Theorem 1. The outcome will be correct, provided that pair-wise comparisons of Δ-values are correct.*

A specific, well known, problem related to numerical inaccuracy is how to represent a straight line in a computer. If a line is represented as a sequence of points, it may look like many short segments with almost the same slope. This would be a disadvantage when using our algorithm as it would generate a large number of segments. Therefore, we suggest to represent slopes explicitly when. The way our algorithm works, numerical errors generated by the representation will be added as we aggregate functions. This allows us to determine an upper bound on the total error, which in practice will be insignificant.

6 Practical Implications

We next demonstrate the algorithm on an example. Even though not too many conclusions can be drawn from a single example, the example probably sheds light over how the algorithm can work in many practical instances.

6.1 A Power Load Management Example

We take an example from power load management. Power load management (or just load management for short) is the concept of managing loads at the customer side in order to run the energy system more efficiently. We here study the technique of direct load management, where a utility has contracts with a number of customers, allowing the utility to control their loads while the customers get something in return. (For a detailed description of the concepts of power load management see, e.g. [Ygge, 1998].) We choose an example which is not too simple (so that it is totally unrealistic) yet not too complex (in order to be understandable). We choose a setting in which a utility has three power plants of different sizes. One has a total capacity of 1MW, and there are two spare plants with the maximum capacity 100kW. The power plants are all assumed to have the same principle shape of the objective functions (though of different scales and with different price levels). The principles of the objective functions are shown in Figure 8. The cost per unit when optimally run (with one or two generators) is \$0.1/kWh for the big plant and \$0.2/kWh for the spare plants.

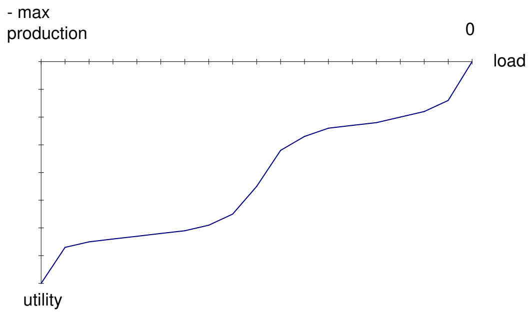

Figure 8: *The characteristics of the power plants.*

The power plants will *produce* resource (i.e. will be allocated a negative amount of resource).[3] The rationale behind the characteristics is that each plant is assumed to have two generators. For small amounts of produced resource, the relative cost is high (i.e. a high increase in cost). At a somewhat higher load the generator runs at its optimal load and the relative cost is low. Then at further higher load the generator is less efficient. At some point the next generator is started and we again get a low relative cost, and a final high relative cost when both generators carry very high load.

At the load side we assume that we can control all loads supplied by the plants at certain costs and benefits. We assume that the utility may cause the indoor temperature to deviate from an agreed on set-point temperature in 500 rooms in some public buildings, such as hospitals and libraries. We further assume that each room can be allocated 0 to 3kW. Deriving the objective functions at the customer side is actually relatively hard, as it includes proper modeling of customer preferences and building characteristics. For this case it is very realistic to assume that the objective functions are concave. The precise numbers is not significant and we let the objective function of every room be $f(r) = 1.25 - (r - 1.5)^2$, where r is in kW. This implies that the utility is optimal at $r = 1.5$ and that at an allocation of 1kW the marginal utility is \$1/kW, i.e. $f'(1) = 1$.

The utility is assumed to have some estimate of future prices and our task is to optimally allocate the resource for the upcoming hour with a resolution of 0.1kW.

We perform the pair-wise aggregations in a tree structure as shown in Figure 9. (By using this type of structure, one can benefit from the computational power inherent in a distribution automation/demand side management system, cf. [Andersson and Ygge, 1998].)

The objective functions of all rooms can be aggregated to a concave objective function. Aggregating the objective functions of the two spare plants results in an aggregated function with 8 segments, see Figure 10. As the number of segments is much smaller than the number of points, the difference in complexity between our algorithm and a general algorithm for non-concave functions is significant.

The objective function resulting from the aggregation in node B is shown in Figure 11. This aggregation results in 9 segments. The two functions to be aggregated at the root now consists of 4 and 9 segments respectively. Also here the improvement in complexity is significant.

[3] Thus, for this example the total amount of resource should sum up to zero. Hence, the range present in, e.g., the complexity analysis, will be the smallest of the amount that can be produced and the amount that can be consumed. Presumably the power utility has some rather accurate knowledge about reasonable allocations, and hence this range can be reduced significantly.

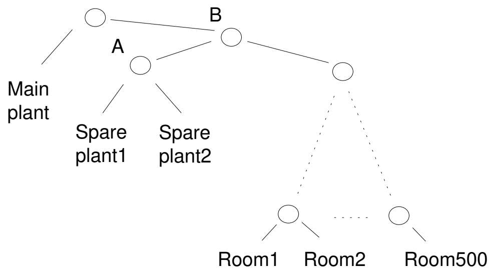

Figure 9: *A suggested tree structure for the load management example.*

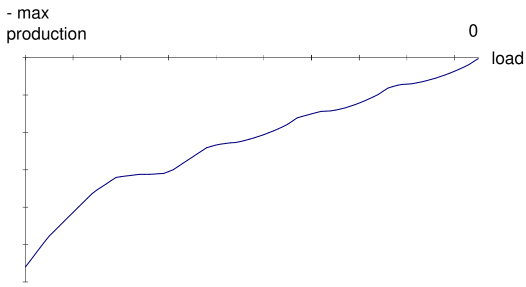

Figure 10: *The result of aggregating the two spare plants, cf. node A in Figure 9.*

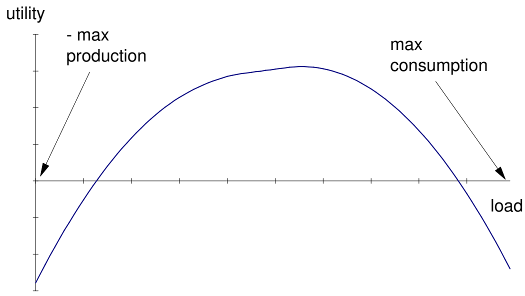

Figure 11: *The result of aggregating all the rooms with the two spare plants, cf. node B in Figure 9.*

6.2 Discussion

If the precision in the above example is increased (i.e. a larger number of points for each function is used), the number of segments remains the same. Hence – for this example – the complexity of our algorithm is linear in the number of points, while the complexity of a traditional algorithm is quadratic. This phenomenon of linear complexity appears whenever the number of segments does not increase with the precision in the representation. We believe this is the case in many applications.

Another phenomenon that could be observed above was the growth of number of segments in the aggregated functions. In the example the number of segments of an aggregated function was the sum of the number of segments in the two functions being aggregated. In our experience this is a typical behavior. However, it is possible to construct examples resulting in an arbitrarily large number of segments in the aggregated function, even when each of the functions being aggregated contains only two concave segments each. Likewise, it is possible to construct examples where both functions being aggregated consist of many segments, but where the aggregated function is simple.

7 Conclusions and Future Work

In this article we introduced a new resource allocation algorithm. It was shown that the number of concave and convex segments can serve as measure for how "well-shaped" objective functions are, and that many instances with non-concave objective functions can be managed efficiently. The main enabling contribution for this result is that all candidates for an optimal solution can be found efficiently (in linear time) not only in two concave segments, but also in two convex segments (which turned out to be relatively simple), and one concave and one convex segment (which is non-trivial).

We believe that our new algorithm will outperform traditional algorithms in a large number of realistic examples. Admittedly there are cases (for example where the objective functions of a large number of nodes has large number of segments) were the algorithm will not perform very well. Even though the asymptotic complexity is never worse than $\mathcal{O}(nR^2)$ (as can be seen from Theorem 1), constant factors are somewhat larger than for a traditional algorithm. Therefore it is wise to – before each aggregation – check whether the number of segments is too large, and if so choose a traditional method instead. This would result in a combined algorithm which in the worst case is only slightly less efficient than traditional methods, but in many practical cases is significantly more efficient.

When arranging the nodes in a tree structure with all nodes as leafs and observing the aggregated objective functions (i.e. the objective functions of the internal nodes of the tree) it is striking how fast the objective functions strives towards objective functions that *very* accurately can be approximated by concave functions. This strongly suggests that even if there is a large number of nodes holding non-concave objective functions it is possible to – when the number of segments becomes too large high up in the tree - reduce the number of segments by approximating two ore more segments with concave segments. Constructing an approximate algorithm based on the algorithm introduced in this article is part of current work.

References

[Andersson and Ygge, 1998] A. Andersson and F. Ygge. Managing large scale computational markets. In H. El-Rewini, editor, *Proceedings of the Software Technology track of the 31th Hawaiian International Conference on System Sciences (HICSS31)*, volume VII, pages 4 – 14. IEEE Computer Society, Los Alamos, January 1998. ISBN 0-8186-8251-5, ISSN 1060-3425, IEEE Catalog Number 98TB100216. (Available from http://www.enersearch.se/ygge).

[Ibaraki and Katoh, 1988] T. Ibaraki and N. Katoh. *Resource Allocation Problems – Algorithmic Approaches.* The MIT Press, 1988.

[Press *et al.*, 1994] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipies in C.* Cambridge University Press, 1994. Second Edition.

[Ygge, 1998] F. Ygge. *Market-Oriented Programming and its Application To Power Load Management.* PhD thesis, Department of Computer Science, Lund University, 1998. ISBN 91-628-3055-4, CODEN LUNFD6/(NFCS-1012)/1-224/(1998) (Available from http://www.enersearch.se/ygge).