Math. Prog. Comp. (2011) 3:1–35
DOI 10.1007/s12532-011-0023-y

FULL LENGTH PAPER

Faster min–max resource sharing in theory and practice

Dirk Müller · Klaus Radke · Jens Vygen

Received: 1 July 2010 / Accepted: 9 February 2011 / Published online: 1 March 2011

Abstract We consider the (block-angular) min–max resource sharing problem, which is defined as follows. Given finite sets $\mathcal{R}$ of resources and $\mathcal{C}$ of customers, a convex set $\mathcal{B}_c$, called block, and a convex function $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$ for every $c \in \mathcal{C}$, the task is to find $b_c \in \mathcal{B}_c$ $(c \in \mathcal{C})$ approximately attaining $\lambda^* := \inf\{\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(b_c))_r \mid b_c \in \mathcal{B}_c \ (c \in \mathcal{C})\}$. As usual we assume that g_c can be computed efficiently and we have a constant $\sigma \geq 1$ and oracle functions $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_c$, called block solvers, which for $c \in \mathcal{C}$ and $y \in \mathbb{R}_+^{\mathcal{R}}$ return an element $b_c \in \mathcal{B}_c$ with $y^\top g_c(b_c) \leq \sigma \inf_{b\in\mathcal{B}_c} y^\top g_c(b)$. We describe a simple algorithm which solves this problem with an approximation guarantee $\sigma(1+\omega)$ for any $\omega > 0$, and whose running time is $O(\theta(|\mathcal{C}|+|\mathcal{R}|)\log|\mathcal{R}|(\log\log|\mathcal{R}|+\omega^{-2}))$ for any fixed $\sigma \geq 1$, where θ is the time for an oracle call. This generalizes and improves various previous results. We also prove other bounds and describe several speed-up techniques. In particular, we show how to parallelize the algorithm efficiently. In addition we review another algorithm, variants of which were studied before. We show that this algorithm is almost as fast in theory, but it was not competitive in our experiments. Our work was motivated mainly by global routing in chip design. Here the blocks are mixed-integer sets (whose elements are associated with Steiner trees), and we combine our algorithm with randomized rounding. We present experimental results on instances

Research was done while K. Radke was at the University of Bonn.

D. Müller (✉) · J. Vygen
Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2,
53113 Bonn, Germany
e-mail: mueller@or.uni-bonn.de

J. Vygen
e-mail: vygen@or.uni-bonn.de

K. Radke
Department of Computer Science, RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany

resulting from recent industrial chips, with millions of customers and resources. Our algorithm solves these instances nearly optimally in less than two hours.

Keywords Min–max resource sharing · Fractional packing · Fully polynomial approximation scheme · Parallelization · Global routing · Chip design

Mathematics Subject Classification (2010) 90C27 · 90C90 · 90C59 · 90C48 · 90C06

1 Introduction

The problem of sharing a set of limited resources between users (customers) in an optimal way is fundamental. The common mathematical model has been called the min–max resource sharing problem. Well-studied special cases are the fractional packing problem and the maximum concurrent flow problem. The only known exact algorithms for these problems use general linear (or convex) programming. Shahrokhi and Matula [20] were the first to design a combinatorial approximation scheme for the maximum concurrent flow problem. Subsequently, this result was improved, simplified, and generalized many times.

This paper is a further step on this line. In particular we provide a simple algorithm and a simple proof of the best performance guarantee in significantly smaller running time. We also describe several techniques that speed up the algorithm in practice and parallelize it. Moreover, we implemented the algorithm and show experimental results for an application to global routing of VLSI chips.

The problem. The (block-angular) MIN–MAX RESOURCE SHARING PROBLEM is defined as follows. Given finite sets $\mathcal{R}$ of *resources* and $\mathcal{C}$ of *customers*, an implicitly given convex set $\mathcal{B}_c$, called *block*, of *feasible solutions* for customer c (for $c \in \mathcal{C}$), and a nonnegative convex function $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$ for $c \in \mathcal{C}$ specifying the *resource consumption*, the task is to find $b_c \in \mathcal{B}_c$ $(c \in \mathcal{C})$ approximately attaining

$$\lambda^* := \inf \left\{ \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))_r \,\middle|\, b_c \in \mathcal{B}_c \,(c \in \mathcal{C}) \right\}, \tag{1}$$

i.e. approximately minimizing the largest resource consumption.

As usual we assume that g_c can be computed efficiently and we have a constant $\sigma \geq 1$ and oracle functions $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_c$, called *block solvers*, which for $c \in \mathcal{C}$ and $y \in \mathbb{R}_+^{\mathcal{R}}$ return an element $b_c \in \mathcal{B}_c$ with $y^\top g_c(b_c) \leq \sigma \operatorname{opt}_c(y)$, where $\operatorname{opt}_c(y) := \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$. Block solvers are called *exact* if $\sigma = 1$ and *strong* if $\sigma > 1$ can be chosen arbitrarily small; otherwise they are called *weak*.

Note that previous authors often required that $\mathcal{B}_c$ is compact, but we do not need this assumption. Some algorithms require *bounded block solvers*: for $c \in \mathcal{C}$, $y \in \mathbb{R}_+^{\mathcal{R}}$, and $\mu > 0$, they return an element $b_c \in \mathcal{B}_c$ with $g_c(b_c) \leq \mu \mathbb{1}$ and $y^\top g_c(b_c) \leq \sigma \inf\{y^\top g_c(b) \mid b \in \mathcal{B}_c,\ g_c(b) \leq \mu \mathbb{1}\}$ (by $\mathbb{1}$ we denote the all-one vector). They can also be exact, strong, or weak.

Table 1 Approximation algorithms for the MIN–MAX RESOURCE SHARING PROBLEM

	Block solver	Number of oracle calls	
Grigoriadis, Khachiyan [8]	Strong, bounded	$O(\|\mathcal{C}\|^2 \log \|\mathcal{R}\|(\log \|\mathcal{C}\| + \omega^{-2}))$	
Grigoriadis, Khachiyan [8]	Strong, bounded	$O(\|\mathcal{C}\| \log \|\mathcal{R}\|(\log \|\mathcal{C}\| + \omega^{-2}))$	(randomized)
Grigoriadis, Khachiyan [9]	Strong, bounded	$O(\|\mathcal{C}\|^2 \log \|\mathcal{R}\|(\log \|\mathcal{R}\| + \omega^{-2}))$	
Grigoriadis, Khachiyan [9]	Strong, unbounded	$O(\|\mathcal{C}\|\|\mathcal{R}\|(\log \|\mathcal{R}\| + \omega^{-2} \log \omega^{-1}))$	
Jansen, Zhang [13]	Strong, unbounded	$O(\|\mathcal{C}\|\|\mathcal{R}\|(\log \|\mathcal{R}\| + \omega^{-2}))$	
Jansen, Zhang [13]	Weak, unbounded	$O(\|\mathcal{C}\|\|\mathcal{R}\|(\log \|\mathcal{R}\| + \omega^{-2} \log \omega^{-1}))$	
Our algorithm (Sect. 4)	Weak, unbounded	$O((\|\mathcal{C}\| + \|\mathcal{R}\|) \log \|\mathcal{R}\|(\log \log \|\mathcal{R}\| + \omega^{-2}))$	
Our algorithm (Sect. 5)	Weak, unbounded	$O(\min\{\|\mathcal{C}\|\rho, \|\mathcal{C}\| + \|\bar{\mathcal{R}}\|\} \log \|\mathcal{R}\|(\log \|\mathcal{R}\| + \omega^{-2})$	
Our algorithm (Sect. 5)	Weak, bounded	$O(\|\mathcal{C}\| \log \|\mathcal{R}\|(\log \|\mathcal{R}\| + \omega^{-2})$	
Young–Khandekar (Sect. 7)	Weak, unbounded	$O((\|\mathcal{C}\| + \|\mathcal{R}\|) \log(\|\mathcal{C}\| + \|\mathcal{R}\|)(\log \log \|\mathcal{R}\| + \omega^{-2})$	

Running times are shown for fixed $\sigma \geq 1$

All algorithms that we consider are fully polynomial approximation schemes relative to σ, i.e. for any given $\omega > 0$ they compute a solution $b_c \in \mathcal{B}_c$ $(c \in \mathcal{C})$ with $\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(b_c))_r \leq \sigma(1+\omega)\lambda^*$, and the running time depends polynomially on ω^{-1}. By θ we denote the time for an oracle call (to a block solver). Moreover, we write $\rho := \max\left\{1, \sup\left\{\frac{(g_c(b))_r}{\lambda^*} \mid r \in \mathcal{R}, c \in \mathcal{C}, b \in \mathcal{B}_c\right\}\right\}$ and $\bar{\mathcal{R}} := \{r \in \mathcal{R} \mid \exists c \in \mathcal{C}, b \in \mathcal{B}_c \text{ with } (g_c(b))_r > \lambda^*\}$.

Previous work. Grigoriadis and Khachiyan [8] were the first to present such an algorithm for the general MIN–MAX RESOURCE SHARING PROBLEM. It makes $O(|\mathcal{C}|^2 \log |\mathcal{R}|(\log |\mathcal{C}| + \omega^{-2}))$ calls to a strong bounded block solver. They also have a randomized version whose expected running time is $|\mathcal{C}|$ times faster. In [9] they proposed an algorithm which needs $O(|\mathcal{C}|^2 \log |\mathcal{R}|(\log |\mathcal{R}| + \omega^{-2}))$ calls to a strong bounded block solver or $O(|\mathcal{C}||\mathcal{R}|(\log |\mathcal{R}| + \omega^{-2} \log \omega^{-1}))$ calls to a strong unbounded block solver.

Jansen and Zhang [13] generalized this and need $O(|\mathcal{C}||\mathcal{R}|(\log |\mathcal{R}| + \omega^{-2} \log \omega^{-1}))$ calls to a weak unbounded block solver. They also showed that $O(|\mathcal{C}||\mathcal{R}|(\log |\mathcal{R}| + \omega^{-2}))$ calls to a strong unbounded block solver suffice.

The first part of Table 1 summarizes the previous results. Khandekar [14] claimed to have an algorithm with $O((|\mathcal{C}| + |\mathcal{R}|) \log(|\mathcal{C}| + |\mathcal{R}|)(\log \log |\mathcal{R}| + \omega^{-2}))$ calls to a strong block solver, but his proof contains two gaps (which we point out in the appendix).

Fractional packing. The special case where the functions g_c $(c \in \mathcal{C})$ are linear is often called the FRACTIONAL PACKING PROBLEM (although sometimes this name is used for different problems). For this special case faster algorithms using unbounded block solvers are known. Plotkin, Shmoys and Tardos [17] require a strong block solver and $O(\omega^{-2}\rho'|\mathcal{C}| \log(|\mathcal{R}|\omega^{-1}))$ oracle calls, where $\rho' := \max\{\sum_{c\in\mathcal{C}} (g_c(b_c))_r | r \in \mathcal{R}, b_c \in \mathcal{B}_c\ (\forall c)\}$. Note that $\rho' \leq \lambda^*|\mathcal{C}|\rho$. They also present a randomized version that is sometimes faster. Charikar et al. [5] subsequently relaxed the unrandomized

result to weak block solvers in the same number of oracle calls. Villavicencio and Grigoriadis [21] analyzed a deterministic variant of the randomized algorithm of Grigoriadis and Khachiyan [8] in the linear case and bounded the number of calls to a strong bounded block solver by $O(|\mathcal{C}| \log |\mathcal{R}|(\log |\mathcal{C}| + \omega^{-2}))$. They claim (without proof) that their result also holds in the nonlinear case. Young's algorithm [23] makes $O((\lambda^*)^{-1}\omega^{-2}\rho'|\mathcal{C}| \log |\mathcal{R}|)$ calls to a weak block solver for any fixed $\sigma \geq 1$. The algorithm of Garg and Könemann [7] needs $O(|\mathcal{C}| \log |\mathcal{R}|(\log |\mathcal{R}| + \omega^{-2}))$ calls to an exact bounded block solver or $O((|\mathcal{C}| + |\mathcal{R}|) \log |\mathcal{R}|(\log |\mathcal{R}| + \omega^{-2}))$ calls to an exact unbounded block solver. Young [24] gives an algorithm for the MIXED PACKING AND COVERING PROBLEM (a generalization of the FRACTIONAL PACKING PROBLEM) and presents a binary search technique to approximately solve the optimization version in $O((|\mathcal{C}| + |\mathcal{R}|) \log(|\mathcal{C}| + |\mathcal{R}|)(\log \log |\mathcal{R}| + \omega^{-2}))$ calls to an exact unbounded block solver.

The book by Bienstock [2] presents the most important algorithms for the linear case and their history, and it also contains interesting computational results.

Bienstock and Iyengar [3] managed to reduce the dependence on ω from $O(\omega^{-2})$ to $O(\omega^{-1})$. Their algorithm does not call a block solver, but requires the linear resource consumption functions to be explicitly specified by a $|\mathcal{R}| \times \dim(\mathcal{B}_c)$-matrix G_c for each $c \in \mathcal{C}$. So their algorithm does not apply to the general problem, but to an interesting special case which includes the MAXIMUM CONCURRENT FLOW PROBLEM. The algorithm solves $O(\omega^{-1}\sqrt{Kn \log |\mathcal{R}|})$ separable convex quadratic programs, where $n := \sum_{c\in\mathcal{C}} \dim(\mathcal{B}_c)$, and $K := \max_{1\leq i\leq|\mathcal{R}|} \sum_{c\in\mathcal{C}} k_i^c$, with k_i^c being the number of nonzero entries in the ith row of G_c.

Our results. We describe a simple algorithm for the general MIN–MAX RESOURCE SHARING PROBLEM. Our algorithm and its analysis use ideas of Garg and Könemann [7], Plotkin, Shmoys and Tardos [17], and Young [24].

With a weak unbounded block solver (the most general case) we obtain a running time of $O(\theta(|\mathcal{C}| + |\mathcal{R}|) \log |\mathcal{R}|(\log \log |\mathcal{R}| + \omega^{-2}))$ for any fixed $\sigma \geq 1$. This generalizes and improves results for the linear case and decreases the best known running time for the general MIN–MAX RESOURCE SHARING PROBLEM significantly (cf. Table 1).

With a weak bounded block solver we also get a running time of $O(\theta|\mathcal{C}| \log |\mathcal{R}|$ $(\log |\mathcal{R}| + \omega^{-2}))$. We also get other bounds, depending on ρ or on $|\bar{\mathcal{R}}|$, that are often better in practice. Moreover, we describe several speed-up techniques that drastically decrease the running time in practice. In particular, we show how our algorithm can be parallelized efficiently without loss of quality.

Moreover, we analyze an algorithm essentially due to Young [24] and generalized by Khandekar [14]. We obtain the same worst-case running time as claimed (but not correctly proved) by Khandekar [14], which is slightly worse than the bound for the other algorithm (cf. Table 1).

The motivation of our practical work is an application to global routing in chip design. Here the blocks are mixed-integer sets whose elements are associated with Steiner trees. We describe a fractional relaxation which is the MIN–MAX RESOURCE SHARING PROBLEM. As block solvers we need an approximation algorithm for Steiner trees in graphs. We also generalize the randomized rounding paradigm and obtain an improved bound.

Finally we present experimental results for instances from current industrial chips, with millions of customers and resources. We show that such problems can be solved within a few percent of optimum in less than 2 h. In our experiments, our algorithm outperformed the Young–Khandekar algorithm.

Structure of the paper. Our paper is organized as follows. In Sect. 2 we review some simple and well-known a priori bounds. In Sect. 3 we present our core algorithm, which is the basis of most of the remaining paper. In Sect. 4 we combine it with scaling and binary search techniques and obtain our main theoretical result, improving the best known running time for the general MIN–MAX RESOURCE SHARING PROBLEM.

In Sect. 5 we present further bounds, some of which are even better for many practical instances. We also show how to reduce the number of oracle calls in practice significantly. In Sect. 6 we parallelize our algorithm without losing our performance guarantees. In Sect. 7 we generalize Young's [24] algorithm, using and correcting ideas of Khandekar [14].

In Sect. 8 we introduce our application: global routing in chip design. We describe the constraints and objectives and observe that a fractional relaxation is our MIN–MAX RESOURCE SHARING PROBLEM. The oracles can be implemented by an approximation algorithm for the Steiner tree problem in graphs. We also describe how randomized rounding works here.

Section 9 contains implementation details that require consideration. Some of these are needed to understand the experimental results, which we present in Sect. 10.

2 Bounds

We first recall two well-known results that yield a priori bounds on the optimum λ^*:

Lemma 1 *Let $y \in \mathbb{R}_+^{\mathcal{R}}$ be some cost vector with $y^\top \mathbb{1} \neq 0$. Then $\frac{\sum_{c\in\mathcal{C}} \operatorname{opt}_c(y)}{y^\top \mathbb{1}} \leq \lambda^*$.*

Proof Let $\delta > 0$ and $(b_c \in \mathcal{B}_c)_{c\in\mathcal{C}}$ a solution with $\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(b_c))_r < (1+\delta)\lambda^*$. Then

$$\frac{\sum_{c\in\mathcal{C}} \operatorname{opt}_c(y)}{y^\top \mathbb{1}} \leq \frac{\sum_{c\in\mathcal{C}} y^\top g_c(b_c)}{y^\top \mathbb{1}} < \frac{(1+\delta)\lambda^* y^\top \mathbb{1}}{y^\top \mathbb{1}} = (1+\delta)\lambda^*.$$

□

Lemma 2 *Let $\lambda^{ub} := \max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(f_c(\mathbb{1})))_r$. Then $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.*

Proof Trivially, $\lambda^* \leq \lambda^{ub}$. Moreover, by Lemma 1 we have

$$\lambda^* \geq \frac{\sum_{c\in\mathcal{C}} \operatorname{opt}_c(\mathbb{1})}{\mathbb{1}^\top \mathbb{1}} \geq \frac{\sum_{c\in\mathcal{C}} \mathbb{1}^\top g_c(f_c(\mathbb{1}))}{\sigma \mathbb{1}^\top \mathbb{1}} = \frac{1}{|\mathcal{R}|\sigma} \sum_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(f_c(\mathbb{1})))_r \geq \frac{\lambda^{ub}}{|\mathcal{R}|\sigma}.$$

□

3 The core algorithm

Our algorithm and its analysis use ideas of Garg and Könemann [7], Plotkin, Shmoys and Tardos [17], and Young [24]. The core algorithm can be described as follows.

RESOURCE SHARING ALGORITHM

Input: An instance of the MIN–MAX RESOURCE SHARING PROBLEM, i.e. finite sets $\mathcal{R}$ and $\mathcal{C}$, and for each $c \in \mathcal{C}$ an oracle function $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_c$ and a convex function $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$ with the property $y^\top g_c(f_c(y)) \leq \sigma \operatorname{opt}_c(y)$ for all $y \in \mathbb{R}_+^{\mathcal{R}}$ and a constant $\sigma \geq 1$. Parameters $\epsilon > 0$ and $t \in \mathbb{N}$.

Output: For each $c \in \mathcal{C}$ a convex combination of vectors in $\mathcal{B}_c$, given by $\sum_{b \in \mathcal{B}_c} x_{c,b} b$. A cost vector $y \in \mathbb{R}_+^{\mathcal{R}}$.

Set $\alpha_r := 0$ and $y_r := 1$ for each $r \in \mathcal{R}$.
Set $X_c := 0$ and $x_{c,b} := 0$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$.
for $p := 1$ **to** t **do**
  while there exists a $c \in \mathcal{C}$ with $X_c < p$ **do**
    Let $c \in \mathcal{C}$ with $X_c < p$.
    ALLOCATERESOURCES(c).
Set $x_{c,b} := \frac{1}{t} x_{c,b}$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$.

Procedure ALLOCATERESOURCES($c \in \mathcal{C}$):

Set $b := f_c(y)$ and $a := g_c(b)$.
Set $\xi := \min\{p - X_c, 1/\max\{a_r \mid r \in \mathcal{R}\}\}$.
Set $x_{c,b} := x_{c,b} + \xi$ and $X_c := X_c + \xi$.
Set $\alpha := \alpha + \xi a$.
foreach $r \in \mathcal{R}$ with $a_r \neq 0$ **do**
  Set $y_r := e^{\epsilon \alpha_r}$.

Of course x is not stored explicitly, but rather by a set of triples $(c, b, x_{c,b})$ for those $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$ for which $x_{c,b} > 0$. We will call the outer iterations ($p = 1, \ldots, t$) of this algorithm *phases*. Let k_p denote the number of inner iterations in phase p. Let $y^{(p,i)}$, $c^{(p,i)}$, $a^{(p,i)}$, and $\xi^{(p,i)}$ denote y, c, a, and ξ at the end of the ith inner iteration within the pth phase ($p = 1, \ldots, t, i = 1, \ldots, k_p$). Let $y^{(p)} := y^{(p,k_p)}$ be the cost vector at the end of phase p. We write $y^{(p,0)} := y^{(p-1)}$ etc., and $y^{(0)}$ is the value of y after initialization. The resource prices y can be interpreted as dual variables, but we do not use duality explicitly.

The RESOURCE SHARING ALGORITHM yields $x_{c,b} \geq 0$ for all $b \in \mathcal{B}_c$ with $\sum_{b \in \mathcal{B}_c} x_{c,b} = 1$. Hence we have a convex combination of vectors in $\mathcal{B}_c$ for each $c \in \mathcal{C}$. The quality of the solution can be estimated as follows.

Lemma 3 *Let* (x, y) *be the output of the* RESOURCE SHARING ALGORITHM, *and let*

$$\lambda_r := \sum_{c \in \mathcal{C}} \left(g_c \left(\sum_{b \in \mathcal{B}_c} x_{c,b} b \right) \right)_r$$

and $\lambda := \max_{r\in\mathcal{R}} \lambda_r$. *Then*

$$\lambda \;\le\; \frac{1}{\epsilon t} \ln\left(\mathbb{1}^\top y\right).$$

Proof Using the convexity of the functions g_c we have for $r \in \mathcal{R}$:

$$\begin{aligned}\lambda_r \le \sum_{c\in\mathcal{C}} \sum_{b\in\mathcal{B}_c} x_{c,b}(g_c(b))_r &= \frac{1}{t}\sum_{p=1}^{t}\sum_{i=1}^{k_p} \xi^{(p,i)}(a^{(p,i)})_r = \frac{1}{t}\alpha_r^{(t)} \\ &= \frac{1}{\epsilon t}\ln y_r^{(t)} \le \frac{1}{\epsilon t}\ln\left(\mathbb{1}^\top y^{(t)}\right).\end{aligned}$$

□

Lemma 4 *Let* $\sigma \ge 1$, *and* $\epsilon > 0$. *Let* $\epsilon' := (e^\epsilon - 1)\sigma$. *If* $\epsilon'\lambda^* < 1$, *then*

$$\mathbb{1}^\top y^{(t)} \;\le\; |\mathcal{R}| e^{t\epsilon'\lambda^*/(1-\epsilon'\lambda^*)}.$$

Proof We will consider the term $\mathbb{1}^\top y^{(p)}$ for all phases p. Initially we have $\mathbb{1}^\top y^{(0)} = |\mathcal{R}|$. We can estimate the increase of the resource prices as follows:

$$\begin{aligned}\sum_{r\in\mathcal{R}} y_r^{(p,i)} &= \sum_{r\in\mathcal{R}} y_r^{(p,i-1)} e^{\epsilon\xi^{(p,i)}(a^{(p,i)})_r} \\ &\le \sum_{r\in\mathcal{R}} y_r^{(p,i-1)} + (e^\epsilon - 1)\sum_{r\in\mathcal{R}} y_r^{(p,i-1)}\xi^{(p,i)}(a^{(p,i)})_r, \qquad (2)\end{aligned}$$

because $\xi^{(p,i)}(a^{(p,i)})_r \le 1$ for $r \in \mathcal{R}$, and $e^x \le 1 + \frac{e^\epsilon - 1}{\epsilon}x$ for $0 \le x \le \epsilon$.

Moreover,

$$\sum_{r\in\mathcal{R}} y_r^{(p,i-1)}(a^{(p,i)})_r \le \sigma \operatorname{opt}_{c^{(p,i)}}(y^{(p,i-1)}). \qquad (3)$$

Using (2), (3), the monotonicity of y, the fact $\sum_{i:c^{(p,i)}=c} \xi^{(p,i)} = 1$ $(\forall c)$, and Lemma 1 we get

$$\begin{aligned}\mathbb{1}^\top y^{(p)} &\le \mathbb{1}^\top y^{(p-1)} + (e^\epsilon - 1)\sigma \sum_{i=1}^{k_p} \xi^{(p,i)} \operatorname{opt}_{c^{(p,i)}}(y^{(p,i-1)}) \\ &\le \mathbb{1}^\top y^{(p-1)} + \epsilon' \sum_{c\in\mathcal{C}} \operatorname{opt}_c(y^{(p)}) \\ &\le \mathbb{1}^\top y^{(p-1)} + \epsilon'\lambda^* \mathbb{1}^\top y^{(p)}\end{aligned}$$

and hence

$$\mathbb{1}^\top y^{(p)} \;\le\; \frac{\mathbb{1}^\top y^{(p-1)}}{1 - \epsilon'\lambda^*}.$$

Combining this with $\mathbb{1}^\top y^{(0)} = |\mathcal{R}|$ and $1 + x \le e^x$ for $x \ge 0$ we get, if $\epsilon'\lambda^* < 1$:

$$\mathbb{1}^\top y^{(t)} \le \frac{|\mathcal{R}|}{(1-\epsilon'\lambda^*)^t} = |\mathcal{R}| \left(1 + \frac{\epsilon'\lambda^*}{1-\epsilon'\lambda^*}\right)^t \le |\mathcal{R}| e^{t\epsilon'\lambda^*/(1-\epsilon'\lambda^*)}.$$

□

Lemma 5 *The number of oracle calls is at most*

$$\min\left\{t\Lambda, t|\mathcal{C}| + \frac{|\mathcal{R}'|}{\epsilon} \ln\left(\mathbb{1}^\top y^{(t)}\right)\right\},$$

where $\Lambda := \sum_{c\in\mathcal{C}} \max\{1, \sup\{g_c(b)_r \mid r \in \mathcal{R}, b \in \mathcal{B}_c\}\}$ *and* $\mathcal{R}' := \{r \in \mathcal{R} \mid \exists c \in \mathcal{C}, b \in \mathcal{B}_c : (g_c(b))_r > 1\}$.

Proof Obviously there are $t|\mathcal{C}|$ inner iterations with $\xi = p - X_c$, and at most $t\Lambda$ inner iterations in total.

Suppose that for $r \in \mathcal{R}'$ there are κ_r inner iterations for which $\xi = 1/(a_c)_r$. Each of these increases y_r by the factor e^ϵ, so $y_r^{(t)} \ge e^{\epsilon\kappa_r}$. Therefore we get

$$\sum_{r\in\mathcal{R}'} \kappa_r \le \frac{1}{\epsilon} \sum_{r\in\mathcal{R}'} \ln y_r^{(t)} \le \frac{|\mathcal{R}'|}{\epsilon} \ln\left(\max_{r\in\mathcal{R}'} y_r^{(t)}\right) \le \frac{|\mathcal{R}'|}{\epsilon} \ln\left(\mathbb{1}^\top y^{(t)}\right).$$

□

4 Applying the core algorithm

Let Λ and $\mathcal{R}'$ be defined as in Lemma 5. As before, θ denotes the time for an oracle call.

Lemma 6 *Given an instance of the* Min–Max Resource Sharing Problem*, we can decide in* $O\left(\theta \log|\mathcal{R}| \min\left\{\Lambda, |\mathcal{C}| + |\mathcal{R}'|\right\}\right)$ *time either that* $\lambda^* > \frac{1}{4\sigma}$ *or that* $\lambda^* \le \frac{1}{2}$.

Proof We set $t := \lceil 18 \ln|\mathcal{R}| \rceil$ and $\epsilon := \frac{1}{3}$. We have $\epsilon' = (e^\epsilon - 1)\sigma \le \frac{2\sigma}{5}$. Then we run the Resource Sharing Algorithm but stop it as soon as $\mathbb{1}^\top y > |\mathcal{R}|e^{t/9}$.

If $\lambda^* \le \frac{1}{4\sigma}$, then $\epsilon'\lambda^* \le \frac{1}{10}$, and Lemma 4 implies $\mathbb{1}^\top y^{(t)} \le |\mathcal{R}|e^{t/9}$. Hence a premature stop implies $\lambda^* > \frac{1}{4\sigma}$.

Otherwise $\mathbb{1}^\top y^{(t)} \le |\mathcal{R}|e^{t/9}$, and Lemma 3 implies $\lambda^* \le \frac{1}{\epsilon t}\ln(\mathbb{1}^\top y^{(t)}) \le \frac{\ln|\mathcal{R}|}{\epsilon t} + \frac{1}{9\epsilon} \le \frac{1}{6} + \frac{1}{3} = \frac{1}{2}$.

As we have $\mathbb{1}^\top y \le |\mathcal{R}|e^{t/9} \le |\mathcal{R}|^3 e$ at any stage, by Lemma 5 the total number of oracle calls is at most $(18\ln|\mathcal{R}| + 1)\min\left\{\Lambda, |\mathcal{C}| + 3|\mathcal{R}'|\right\}$. □

Lemma 7 *Let* $0 < \delta, \delta' < 1$. *Given an instance of the* Min–Max Resource Sharing Problem *with* $\lambda^* \le 1$*, we can compute a* $(\sigma(1+\delta) + \frac{\delta'}{\lambda^*})$*-approximate solution in* $O(\theta \log|\mathcal{R}| \min\{\Lambda, |\mathcal{C}| + |\mathcal{R}'|\sigma\}(\delta\delta')^{-1}\sigma)$ *time.*

Proof We choose $\epsilon := \frac{\delta}{3\sigma}$ and $t := \lceil \frac{\ln |\mathcal{R}|}{\epsilon\delta'} \rceil$. We have $\epsilon \le \frac{1}{3}, e^{\frac{1}{3}} < \frac{7}{5}$, and thus $e^{\epsilon} - 1 \le \epsilon(1 + \frac{3}{5}\epsilon)$. We get $\epsilon' = (e^{\epsilon} - 1)\sigma \le \epsilon(1 + \frac{3}{5}\epsilon)\sigma \le \frac{2\delta}{5} \le \frac{2}{5}$ and $\frac{1}{1-\epsilon'\lambda^*} \le \frac{1}{1-\epsilon'} \le 1 + \frac{5}{3}\epsilon' \le 1 + \frac{2\delta}{3}$ and $\frac{\epsilon'}{\epsilon(1-\epsilon'\lambda^*)} \le \sigma(1+\frac{3}{5}\epsilon)(1+\frac{2\delta}{3}) \le \sigma(1+\frac{\delta}{5}+\frac{2\delta}{3}+\frac{2\delta^2}{15}) \le \sigma(1+\delta)$.

We run the RESOURCE SHARING ALGORITHM and obtain a solution x. As $\epsilon'\lambda^* < 1$, Lemma 3 and Lemma 4 yield:

$$\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} \left(g_c \left(\sum_{b\in\mathcal{B}_c} x_{c,b} b \right) \right)_r \le \frac{1}{\epsilon t} \left(\ln |\mathcal{R}| + \frac{t\epsilon'\lambda^*}{1-\epsilon'\lambda^*} \right) \le \delta' + \frac{\epsilon'}{\epsilon(1-\epsilon'\lambda^*)}\lambda^* \le \delta' + \sigma(1+\delta)\lambda^*.$$

Using Lemma 5 and Lemma 4 we obtain that the number of oracle calls is at most

$$\begin{aligned} &\min \left\{ t\Lambda, t|\mathcal{C}| + \frac{|\mathcal{R}'|}{\epsilon} \ln(\mathbb{1}^\top y^{(t)}) \right\} \\ &\le \min \left\{ t\Lambda, t|\mathcal{C}| + \frac{|\mathcal{R}'|}{\epsilon} \left(\ln |\mathcal{R}| + t\epsilon\lambda^* \frac{\epsilon'}{\epsilon(1-\epsilon'\lambda^*)} \right) \right\} \\ &\le \left\lceil \frac{\ln |\mathcal{R}|}{\epsilon\delta'} \right\rceil \min \left\{ \Lambda, |\mathcal{C}| + |\mathcal{R}'|\delta' + |\mathcal{R}'|\lambda^*\sigma(1+\delta) \right\} \\ &= O\left(\frac{\sigma \ln |\mathcal{R}|}{\delta\delta'} \min \left\{ \Lambda, |\mathcal{C}| + |\mathcal{R}'|\sigma \right\} \right). \end{aligned}$$

□

Together with binary search we get a fully polynomial approximation scheme relative to σ:

Theorem 8 *Given an instance of the* MIN–MAX RESOURCE SHARING PROBLEM *with σ-optimal block solvers for some $\sigma \ge 1$. Then a $\sigma(1+\omega)$-approximate solution can be computed in $O(\theta \log |\mathcal{R}|((|\mathcal{C}| + |\mathcal{R}|) \log\log |\mathcal{R}| + (|\mathcal{C}| + |\mathcal{R}|\sigma)\sigma\omega^{-2}))$ time.*

Proof We first compute $f_c(\mathbb{1})$ for all $c \in \mathcal{C}$ and set $\lambda^{ub} := \max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(f_c(\mathbb{1})))_r$. W.l.o.g. $\lambda^{ub} > 0$ since otherwise we already have an optimum solution. By Lemma 2 we have $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \le \lambda^* \le \lambda^{ub}$.

For $j \in \{0, \ldots, \lfloor \log_2 |\mathcal{R}| \rfloor\}$ we define a new instance $I^{(j)}$ by setting $g_c^{(j)}(b) := 2^{j-1} g_c(b)/\lambda^{ub}$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$. Let $\lambda^{*(j)}$ be the optimum of the instance $I^{(j)}$. We have $\lambda^{*(0)} \le \frac{1}{2}$ and $\lambda^{*(\lfloor \log_2 |\mathcal{R}| \rfloor)} > \frac{1}{4\sigma}$. By binary search and $O(\log\log |\mathcal{R}|)$ applications of Lemma 6 we obtain an index j with $\lambda^{*(j)} \le \frac{1}{2}$ and $\lambda^{*(j+1)} > \frac{1}{4\sigma}$. We have $\frac{1}{4\sigma} < \lambda^{*(j+1)} \le 1$.

We finally apply Lemma 7 to $I^{(j+1)}$ with $\delta := \frac{\omega}{2}$ and $\delta' := \frac{\omega}{8}$. We get an approximation guarantee of $\sigma(1+\delta) + \delta'/\lambda^{*(j+1)} < \sigma(1+\frac{\omega}{2}) + \frac{4\sigma\omega}{8} = \sigma(1+\omega)$.

The total running time is dominated by applying Lemma 6 $O(\log\log |\mathcal{R}|)$ times and Lemma 7 once with $\delta = \frac{\omega}{2}$ and $\delta' = \frac{\omega}{8}$, and is hence as claimed. □

For every fixed $\sigma \ge 1$ the running time is $O(\theta \log |\mathcal{R}|(|\mathcal{C}| + |\mathcal{R}|)(\log\log |\mathcal{R}| + \omega^{-2}))$.

5 Improvements in practice

Theorem 8 gives a much better worst-case running time for the general MIN–MAX RESOURCE SHARING PROBLEM than was known before. In practice this can often be improved even further, as we will discuss in this and the next section.

Theoretical bounds. In the application of Lemma 7 at the end of the proof of Theorem 8 we can actually do better for many practical instances. Let $\rho := \max\{1, \sup\{(g_c(b))_r/\lambda^* \mid r \in \mathcal{R}, c \in \mathcal{C}, b \in \mathcal{B}_c\}\}$ and $\bar{\mathcal{R}} := \{r \in \mathcal{R} \mid \exists c \in \mathcal{C}, b \in \mathcal{B}_c \text{ with } (g_c(b))_r > \lambda^*\}$. These are often small in practice.

Theorem 9 *Given an instance of the* MIN–MAX RESOURCE SHARING PROBLEM *with σ-optimal block solvers for some $\sigma \geq 1$. Then a $\sigma(1+\omega)$-approximate solution can be computed in $O(\theta \log|\mathcal{R}|((|\mathcal{C}|+|\mathcal{R}|)\log\log|\mathcal{R}| + \min\{\rho|\mathcal{C}|, |\mathcal{C}|+|\bar{\mathcal{R}}|\sigma\}\sigma\omega^{-2}))$ time.*

Proof Note that we have $\Lambda \leq \rho|\mathcal{C}|$ and $\mathcal{R}' \subseteq \bar{\mathcal{R}}$ in the application of Lemma 7 at the end of the proof of Theorem 8, because

$$(g_c^{(j+1)}(b))_r = \frac{2^j (g_c(b))_r}{\lambda^{ub}} = \frac{(g_c(b))_r \lambda^{*(j+1)}}{\lambda^*} \leq \frac{(g_c(b))_r}{\lambda^*} \leq \begin{cases} \rho & \text{if } r \in \bar{\mathcal{R}} \\ 1 & \text{if } r \in \mathcal{R}\backslash\bar{\mathcal{R}}. \end{cases}$$

□

By scanning the values of j one by one we also get the following bounds.

Theorem 10 *Given an instance of the* MIN–*Max Resource Sharing Problem with σ-optimal block solvers for some $\sigma \geq 1$. Then a $\sigma(1+\omega)$-approximate solution can be computed in $O(\theta \log|\mathcal{R}|(\min\{\rho|\mathcal{C}|, |\mathcal{C}|+|\bar{\mathcal{R}}|\}\log|\mathcal{R}| + \min\{\rho|\mathcal{C}|, |\mathcal{C}|+|\bar{\mathcal{R}}|\sigma\}\sigma\omega^{-2}))$ time.*

Proof We use the same notation as in the proof of Theorem 8. Instead of binary search, we scan the possible values of j one by one. For $j = 0$ we know that $\lambda^{*(0)} \leq \frac{1}{2}$. While $\lambda^{*(j)} \leq \frac{1}{2}$, we increment j by one and apply Lemma 6. Eventually, after at most $\lfloor \log|\mathcal{R}| \rfloor$ iterations, we obtain an index j with $\lambda^{*(j+1)} > \frac{1}{4\sigma}$ and $\lambda^{*(j)} \leq \frac{1}{2}$, and finally apply Lemma 7 as above.

Note that we have $\Lambda \leq \rho|\mathcal{C}|$ and $\mathcal{R}' \subseteq \bar{\mathcal{R}}$ every time because $\lambda^{*(j)} \leq 1$ whenever we consider index j. The total running time is dominated by applying Lemma 6 $O(\log|\mathcal{R}|)$ times and Lemma 7 once with $\delta = \frac{\omega}{2}$ and $\delta' = \frac{\omega}{8}$, and is hence as claimed. □

Corollary 11 *Given an instance of the* MIN–MAX RESOURCE SHARING PROBLEM *with σ-optimal bounded block solvers for some $\sigma \geq 1$. Then a $\sigma(1+\omega)$-approximate solution can be computed in $O(\theta|\mathcal{C}|\log|\mathcal{R}|(\log|\mathcal{R}| + \omega^{-2}\sigma))$ time.*

Proof In the proof of Theorem 10 the algorithm is called only for instances $I^{(j)}$ where we know $\lambda^{*(j)} \leq 1$. Hence we can restrict $\mathcal{B}_c$ to $\mathcal{B}_c^{(j)} := \{b \in \mathcal{B}_c \mid g_c^{(j)}(b) \leq \mathbb{1}\}$ in iteration j without changing the optimum. This means that we can choose $\Lambda = |\mathcal{C}|$. With the bounded block solver we can optimize over $\mathcal{B}_c^{(j)}$. □

Stopping early. In practice (e.g., in the application described in Sect. 8) we are often only interested in a solution if $\lambda^* \leq 1$. If $\epsilon' < 1$, we can then stop the RESOURCE SHARING ALGORITHM when $\mathbb{1}^\top y^{(p)} > |\mathcal{R}| e^{p\epsilon'/(1-\epsilon')}$ at any stage in phase p, because this implies $\lambda^* > 1$ by Lemma 4.

Reuse of previous solutions. We now show how to reduce the number of oracle calls. For each $c \in \mathcal{C}$ we assume to have a vector $l_c \in \mathbb{R}_+^{\mathcal{R}}$ which for each resource $r \in \mathcal{R}$ gives a lower bound on the amount of r that has to be used to satisfy customer c, i.e. $(l_c)_r \leq (g_c(b))_r$ for each $b \in \mathcal{B}_c$ and $r \in \mathcal{R}$. We can set $(l_c)_r := 0$ for all c and r, but better lower bounds improve running times in practice.

We propose a block solver that calls our given σ-approximate block solver only if the previous result is not good enough anymore. More precisely, we introduce a new parameter $\tau \geq 1$ and new variables $b_c \in \mathcal{B}_c$, $z_c \in \mathbb{R}_+$, and $\hat{y}_c \in \mathbb{R}_+^{\mathcal{R}}$ for $c \in \mathcal{C}$, and replace the first line of ALLOCATERESOURCES by:

if $X_c = 0$ **or** $y^\top g_c(b_c) > \tau(z_c + \sigma(y - \hat{y}_c)^\top l_c)$ **then**
  Set $b_c := f_c(y)$, $z_c := y^\top g_c(b_c)$, and $\hat{y}_c := y$.
Set $b := b_c$ and $a := g_c(b)$.

To show that this behaves like a $\sigma\tau$-approximate block solver, we observe that we have a near-optimum solution even in iterations in which we do not call the block solver:

Lemma 12 *For $p = 1, \ldots, t$ and $i = 1, \ldots, k_p$ we have*

$$(y^{(p,i-1)})^\top a^{(p,i)} \leq \sigma\tau \operatorname{opt}_{c^{(p,i)}}(y^{(p,i-1)}).$$

Proof We show that

$$(y^{(p,i-1)})^\top a^{(p,i)} \leq \tau\left(z_c^{(p,i)} + \sigma(y^{(p,i-1)} - \hat{y}_c^{(p,i)})^\top l_c\right) \tag{4}$$

and

$$z_c^{(p,i)} + \sigma(y^{(p,i-1)} - \hat{y}_c^{(p,i)})^\top l_c \leq \sigma \operatorname{opt}_c(y^{(p,i-1)}), \tag{5}$$

where $c := c^{(p,i)}$, and $z_c^{(p,i)}$ and $\hat{y}_c^{(p,i)}$ are the values of z_c and $\hat{y}_c$, respectively, in the ith iteration of phase p. Both inequalities are evident if the oracle f_c is called in iteration i of phase p and $z_c^{(p,i)} = (y^{(p,i-1)})^\top g_c(f_c(y^{(p,i-1)})) = (y^{(p,i-1)})^\top a^{(p,i)}$ and $\hat{y}_c^{(p,i)} = y^{(p,i-1)}$. Inequality (4) also holds in the other case, in which b_c, z_c, and $\hat{y}_c$ do not change. Inequality (5) holds because z_c and $\hat{y}_c$ have remained constant since the previous iteration (p', i') with $c^{(p',i')} = c$, the y-variables have only increased, and $(l_c)_r \leq (g_c(b))_r$ for each $b \in \mathcal{B}_c$ and $r \in \mathcal{R}$. □

6 Parallelization

Although the proof of Lemma 4 uses a total order of oracle calls in a phase, it does not require a particular order. We can thus perform inner iterations of the RESOURCE SHARING ALGORITHM in parallel and analyze the increase of resource prices in a total order determined by the algorithm. The intention is that if this order provides approximate time stamps for resource price updates, even without coordination between

concurrently performed block solver calls and resource price updates, a σ-optimal solution w.r.t. observed resource prices y' is in most cases $\sigma\tau$-optimal w.r.t. resource prices that would result from applying results in the determined order, for a small $\tau \geq 1$. In order to maintain our performance guarantee we need to check whether the solutions satisfy this condition and otherwise reject them and reschedule the corresponding customers for sequential recomputation at the end of the phase. As we allow resource prices to change during the computations of a block solver, we require *volatility-tolerant block solvers*:

Definition 13 Let $c \in \mathcal{C}$ and $\texttt{Solver}_c$ be a procedure that gets a resource price oracle $H : \mathcal{R} \to \mathbb{R}_+$ as input and produces a pair $(b, z) \in \mathcal{B}_c \times \mathbb{R}_+$ as output. $\texttt{Solver}_c$ is called a *volatility-tolerant* block solver if for any call to it the following condition is satisfied:

For $r \in \mathcal{R}$ let Y_r be the set of resource prices sampled by calling $H(r)$ in this execution of $\texttt{Solver}_c$. Then there is a $y'_r \in Y_r$ for each $r \in \mathcal{R}$ with $Y_r \neq \emptyset$ such that for each vector $y \in \mathbb{R}_+^{\mathcal{R}}$ with $y_r = y'_r$ for $r \in \mathcal{R}$ with $Y_r \neq \emptyset$ we have $z = y^\top g_c(b)$ and $z \leq \sigma \operatorname{opt}_c(y)$.

Note that this implies $(g_c(b))_r = 0$ for each $r \in \mathcal{R}$ with $Y_r = \emptyset$. Trivially, volatility-tolerant block solvers can be implemented by sampling the price of each resource once and then calling a standard block solver which gets the resulting static vector as input. Often the first step can be avoided: For example, Dijkstra's shortest path algorithm can be implemented such that the price of each edge is queried at most once.

We use a shared-memory model in which communication of variable updates is not done explicitly, but implicitly by letting all processors concurrently access the same variables. This model is becoming more and more standard in hardware, with a continuously growing number of processors jointly using the same memory. A particular difficulty arises from the different latencies with which processors observe changes of values stored in memory. An implication of this is that the effects of write operations are not necessarily observed in the same order on different processors (see e.g. [16]). So-called *barrier operations*, which we shall define below, are used to enforce ordering constraints.

To this end, let $\mathcal{O}^r$ be the set of read operations performed in a program execution, and $\mathcal{O}^w$ the set of write operations. We can assume that each such operation reads or writes a single variable and does so *atomically*, i.e. is carried out in a single step and not interrupted by other operations on the same variable. This is justified by the specifications of current microprocessors for properly aligned variables of machine word size, see e.g. [12]. We thus have a relation $\preceq_{\text{mem}}$ on $\mathcal{O}^w$ which is a total order for write operations on the same variable, and $w_1 \not\preceq_{\text{mem}} w_2$ if $w_1, w_2 \in \mathcal{O}^w$ are operations on different variables. Note that different runs on the same input can lead to different relations $\preceq_{\text{mem}}$.

For an operation o_2 that is executed after o_1 on the same processor, we write $o_1 \prec_{\text{exec}} o_2$. We write $o_1 \not\prec_{\text{exec}} o_2$ if o_1 and o_2 are performed on different processors. We further assume to have a mapping $rf : \mathcal{O}^r \to \mathcal{O}^w$ which for each read operation $r \in \mathcal{O}^r$ specifies the write operation $w \in \mathcal{O}^w$ whose result is read by r.

Definition 14 Let $r_1, r_2 \in \mathcal{O}^r$ with $r_1 \prec_{\text{exec}} r_2$, $w_1, w_2 \in \mathcal{O}^w$ with $w_1 \prec_{\text{exec}} w_2$, and w_i and r_j operate on the same variable for $i, j \in \{1, 2\}, i \neq j$. A pair b, b' of *memory*

barrier operations with $w_1 \prec_{\text{exec}} b \prec_{\text{exec}} w_2$ and $r_1 \prec_{\text{exec}} b' \prec_{\text{exec}} r_2$ guarantees that

$$w_2 \preceq_{\text{mem}} rf(r_1) \Rightarrow w_1 \preceq_{\text{mem}} rf(r_2).$$

Instruction reordering, pipelining and speculative execution can also be taken into account. We do not go further into details here and refer to [10] for a thorough introduction to modern processor architectures.

In each phase, our PARALLEL RESOURCE SHARING ALGORITHM calls the subroutine PARALLELALLOCATERESOURCES shown below using any number of processors in parallel, until $X_c + \bar{X}_c = p$ for all $c \in \mathcal{C}$, where $\bar{X}_c$ records rejected solutions. This is done such that no two processors work on the same customer concurrently. Then it sets $\bar{X}_c := 0$ for each $c \in \mathcal{C}$ and repeats the same step using only one processor.

We use a new global variable n for assigning a unique time stamp to each call of PARALLELALLOCATERESOURCES in a phase, and a vector $\alpha' \in \mathbb{R}_+^{\mathcal{R}}$ for storing temporary resource reservations. We set $\alpha'_r := 0$ for each $r \in \mathcal{R}$ in initialization, and $n := 0$ at the beginning of each phase. Our parallel algorithm uses two resource price oracles H_α and $H_{\alpha'}$ which return $e^{\epsilon x}$ for an observed value $x = \alpha_r$ or $x = \alpha'_r$, respectively, for given $r \in \mathcal{R}$. Arithmetic operations on n and the entries of α and α' are performed atomically, i.e. reading a variable, determining its new value and writing the result back is not interleaved with other operations on the same variable. This can be achieved without *locking* operations by using processor instructions such as *Compare-and-Swap*, which are commonly available on all modern microprocessors.

Procedure PARALLELALLOCATERESOURCES($c \in \mathcal{C}$):

Set $(b, z) := \texttt{Solver}_c(H_\alpha)$, $a := g_c(b)$.
Set $\xi := \min\{p - X_c, 1/\max\{a_r \mid r \in \mathcal{R}\}\}$.
Set $\alpha' := \alpha' + \xi a$.
memory_barrier.
Set $n := n + 1$.
memory_barrier.
Set $\bar{y}_r := H_{\alpha'}(r)$ for $r \in \mathcal{R}$ with $a_r > 0$.
memory_barrier.
Set $\bar{y}_r := \bar{y}_r H_\alpha(r) e^{-\epsilon \xi a_r}$ for $r \in \mathcal{R}$ with $a_r > 0$.
if $\bar{y}^\top a \leq \tau z$ **then**
  Set $x_{c,b} := x_{c,b} + \xi$ and $X_c := X_c + \xi$. (successful, accept solution)
  Set $\alpha := \alpha + \xi a$.
  memory_barrier.
else
  Set $\bar{X}_c := \bar{X}_c + \xi$. (reject solution)
Set $\alpha' := \alpha' - \xi a$.

We call each execution of PARALLELALLOCATERESOURCES an inner iteration and define the value to which it increments n as its *time stamp*. We denote $J^{(p)} \subseteq \mathbb{N}$ the time stamps of successful inner iterations in phase p, and $a^{(p,i)}$ and $\xi^{(p,i)}$ the vector a and value ξ, respectively, in the inner iteration of phase p with time stamp $n = i$. Proving the approximation guarantee of the PARALLEL RESOURCE SHARING ALGORITHM relies on the following fact:

Lemma 15 *Let* $1 \leq p \leq t, i \in J^{(p)}, \alpha^{(p,i)} := \alpha^{(p-1)} + \sum_{j \in J^{(p)}: j<i} \xi^{(p,j)} a^{(p,j)}$, *and* $y_r^{(p,i)} := e^{\epsilon \alpha_r^{(p,i)}}$ *for* $r \in \mathcal{R}$. *Let* $y' \in \mathbb{R}_+^{\mathcal{R}}$ *be resource prices observed by* H_α *in line* 1 *in the inner iteration of phase* p *with time stamp* $n = i$, *with* $y'_r := 0$ *if not queried for* $r \in \mathcal{R}$, *and such that* $(y')^\top a = z$. *Further, let* y''_r *be the value of* $\bar{y}_r$ *at the end of this iteration if* $a_r^{(p,i)} > 0$, *and* $y''_r := \infty$ *otherwise. Then* $y' \leq y^{(p,i)} \leq y''$.

Proof To show $y' \leq y^{(p,i)}$, let $r \in \mathcal{R}$ be a resource for which $H_\alpha(r)$ was called in line 1 in iteration i of phase p, and l the read operation on α_r performed by such a call, sampling the value $x \in \mathbb{R}_+$ with $e^{\epsilon x} = y'_r$. If there is no such resource, we are done. Otherwise assume that $y'_r > y_r^{(p,i)}$. By the monotonicity of the exponential function, there is a write operation w with $w \preceq_{\text{mem}} rf(l)$ performed in an iteration $j \in J^{(p)}$ of phase p with $j > i$. Let b' and b denote the memory barrier operations performed in line 4 of iteration i and in line 6 of iteration j, respectively. Let further w' be the write operation that changes the value of n to j in phase p, and l' the read operation on n when incrementing it to i. Since $i < j$, we have $rf(l') \prec_{\text{mem}} w'$. Since $w' \prec_{\text{exec}} b \prec_{\text{exec}} w$ and $l \prec_{\text{exec}} b' \prec_{\text{exec}} l'$, we must have $rf(l) \prec_{\text{mem}} w$ by Definition 14, which is a contradiction.

To show $y^{(p,i)} \leq y''$, let $c \in \mathcal{C}$ be the customer processed in iteration i of phase p, and $r \in \mathcal{R}$ with $a_r^{(p,i)} > 0$. If there is no such resource, we are done. Otherwise let l be the read operation performed by H_α on α_r in line 9, and l' the read operation performed by $H_{\alpha'}$ on α'_r in line 7, sampling values x and x', respectively. Let $j \in J^{(p)}$ with $j < i$ and $a_r^{(p,j)} > 0$. If there is no such j, we are done. Otherwise, let w'_+ the write operation on α'_r in line 3, w'_- the write operation on α'_r in the last line, and w the write operation on α_r in iteration j. Similar to above, by Definition 14 and the barrier operations in lines 4 and 6, we can deduce $w'_+ \preceq_{\text{mem}} rf(l')$. If $w'_- \preceq_{\text{mem}} rf(l')$, we also have $w \preceq_{\text{mem}} rf(l)$ because of the barrier operations in line 8 and after the update of α. Hence we have $x + x' \geq \alpha_r^{(p,i)}$, and the claim follows. □

By Lemma 15, in each successful iteration $i \in J^{(p)}$ $(1 \leq p \leq t)$ we have

$$(y^{(p,i)})^\top a^{(p,i)} \leq (y'')^\top a^{(p,i)} \leq \tau z \leq \sigma \tau \text{opt}_c(y') \leq \sigma \tau \text{opt}_c(y^{(p,i)}) \leq \sigma \tau \text{opt}_c(y^{(p)}).$$

As a corollary, Lemma 4 and thus Theorem 8 also hold for the PARALLEL RESOURCE SHARING ALGORITHM (note that the worst-case bound for the number of oracle calls does not increase by more than a constant factor). In practice, very good speedup values can be obtained by this parallelization approach, as is shown in Sect. 10. We also combined it with the reuse of solutions from earlier iterations discussed above.

7 The Young–Khandekar algorithm

In this section we review an algorithm for min–max resource sharing that is essentially due to Young and Khandekar. Young [24] gave an algorithm for the mixed packing and covering problem using exact oracles. It was later extended to min–max resource sharing with strong oracles and some generalizations of it by Khandekar [14]. Unfortunately, Khandekar's analysis has two flaws (as pointed out in the appendix). Young

and Khandekar applied Fleischer's [6] round-robin technique which exploits the fact that the global approximation guarantee is not affected if the block solutions are just "slightly" approximately optimal. Now if we always had to choose a customer whose block solution has minimum cost among all customers, then we can avoid calling every customer's oracle and instead stay with a customer as long as its block solutions are close enough to this minimum. In our case the minimum grows monotonically and we cannot know its value in advance so we maintain a lower bound to compare with. This leads to a phase-based mechanism where the customers are processed cyclically and the lower bound is raised from phase to phase. One of the errors in Khandekar's work was an inaccurate rise of the lower bound when using approximate oracles.

The following algorithm is closer to Young's original one and works also with weak block solvers. It requires as additional input a target value $\lambda \in \mathbb{R}_+$ and the approximation guarantee σ of the block solvers.

YOUNG–KHANDEKAR ALGORITHM

Input: An instance of the MIN–MAX RESOURCE SHARING PROBLEM, i.e. finite sets $\mathcal{R}$ and $\mathcal{C}$, and for each $c \in \mathcal{C}$ an oracle function $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_c$ and a convex function $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$. A number $\sigma \geq 1$ with the property $y^\top g_c(f_c(y)) \leq \sigma \operatorname{opt}_c(y)$ for all $c \in \mathcal{C}$ and $y \in \mathbb{R}_+^{\mathcal{R}}$. Parameters $\lambda \in \mathbb{R}_+$ and $\epsilon \in (0, \frac{1}{2}]$.

Output: For each $c \in \mathcal{C}$ a convex combination of vectors in $\mathcal{B}_c$ given by $\sum_{b \in \mathcal{B}_c} x_{c,b} b$, or "infeasible".

Set $y_r := 1$ for each $r \in \mathcal{R}$ and $z_c := 1$ for each $c \in \mathcal{C}$.
Set $X_c := 0$ and $x_{c,b} := 0$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$.
Set $\mathcal{A} := \mathcal{C}$.
Set $\Xi := \frac{3}{\epsilon^2}(1 + e^\epsilon \ln |\mathcal{C}| + \ln |\mathcal{R}|)$.
repeat
  Set $\hat{\Phi} := \frac{\mathbb{1}^\top y}{\sum_{c \in \mathcal{A}} z_c}$.
  foreach $c \in \mathcal{A}$ **do**
    while $c \in \mathcal{A}$ and $y^\top g_c(f_c(y)) \leq (1+\epsilon)\sigma\lambda z_c \hat{\Phi}$ **do**
      Set $b := f_c(y)$ and $a := g_c(b)$.
      Set $\bar{\sigma} := \frac{y^\top a}{\lambda z_c \hat{\Phi}}$.
      Set $\xi := \frac{1}{\max\{\max_{r \in \mathcal{R}} a_r, \bar{\sigma}\lambda\}}$.
      Set $x_{c,b} := x_{c,b} + \xi$ and $X_c := X_c + \xi$.
      Set $z_c := z_c \cdot e^{-\epsilon\xi\bar{\sigma}\lambda}$.
      Set $y_r := y_r \cdot e^{\epsilon\xi a_r}$ for each $r \in \mathcal{R}$.
      if $z_c \leq e^{-\epsilon\Xi}$ **then** set $\mathcal{A} := \mathcal{A} \backslash \{c\}$.
  if $\mathcal{A} \neq \emptyset$ and $\frac{\mathbb{1}^\top y}{\sum_{c \in \mathcal{A}} z_c} \leq (1+\epsilon)\hat{\Phi}$ **then stop**, **return** "infeasible".
until $\mathcal{A} = \emptyset$.
Set $x_{c,b} := \frac{1}{X_c} x_{c,b}$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$.

We call the iterations of the outer loop *phases* and the iterations of the innermost loop *steps*. In the following we will again use the variables with a superscript in parentheses to indicate their value at a special time. Here (p, c, s) means the end of the sth step of customer c within the pth phase, and (p) means the end of the pth phase. Furthermore we denote the number of steps for customer c in the pth phase with $s_c^{(p)}$.

Lemma 16 *If the algorithm returns "infeasible", then* $\lambda^* > \lambda$.

Proof At the end of a phase we have $y^\top g_c(f_c(y)) > (1+\epsilon)\sigma\lambda z_c\hat{\Phi}$ for each $c \in \mathcal{A}$. As the oracles are σ-approximate we also have $\sigma \operatorname{opt}_c(y) \geq y^\top g_c(f_c(y))$ and hence

$$\operatorname{opt}_c(y) > (1+\epsilon)\lambda z_c\hat{\Phi} \qquad (\forall c \in \mathcal{A}).$$

The algorithm returns "infeasible" only if $(1+\epsilon)\hat{\Phi} \geq \frac{\mathbb{1}^\top y}{\sum_{c\in\mathcal{A}} z_c}$. With Lemma 1 this implies

$$\lambda^* \geq \sum_{c\in\mathcal{C}} \frac{\operatorname{opt}_c(y)}{\mathbb{1}^\top y} \geq \sum_{c\in\mathcal{A}} \frac{\operatorname{opt}_c(y)}{\mathbb{1}^\top y} > \sum_{c\in\mathcal{A}} \frac{(1+\epsilon)\lambda z_c\hat{\Phi}}{\mathbb{1}^\top y} \geq \sum_{c\in\mathcal{A}} \frac{\lambda z_c}{\sum_{c\in\mathcal{A}} z_c} = \lambda.$$

□

Lemma 17 *In every phase* p *we have*

$$\ln\left(\frac{\mathbb{1}^\top y^{(p)}}{\mathbb{1}^\top y^{(p-1)}}\right) \leq e^\epsilon \cdot \ln\left(\frac{\sum_{c\in\mathcal{A}^{(p-1)}} z_c^{(p-1)}}{\sum_{c\in\mathcal{A}^{(p-1)}} z_c^{(p)}}\right).$$

Proof First we bound the growth of the prices in a step. By the definition of ξ we can apply the two inequalities $e^{\epsilon x} \leq 1 + (e^\epsilon - 1)x$ and $x \leq \frac{1}{1-e^{-\epsilon}}(1 - e^{-\epsilon x})$ for $0 \leq x \leq 1$ in the following.

$$\begin{aligned}
\mathbb{1}^\top y^{(p,c,s)} &= \sum_{r\in\mathcal{R}} y_r^{(p,c,s-1)} \cdot e^{\epsilon\xi^{(p,c,s)}(a^{(p,c,s)})_r} \\
&\leq \sum_{r\in\mathcal{R}} y_r^{(p,c,s-1)} \left(1 + (e^\epsilon - 1)\xi^{(p,c,s)}(a^{(p,c,s)})_r\right) \\
&= \mathbb{1}^\top y^{(p,c,s-1)} + (e^\epsilon - 1)\xi^{(p,c,s)} y^{(p,c,s-1)^\top} a^{(p,c,s)} \\
&= \mathbb{1}^\top y^{(p,c,s-1)} + (e^\epsilon - 1)\xi^{(p,c,s)}\bar{\sigma}^{(p,c,s)}\lambda z_c^{(p,s-1)}\hat{\Phi}^{(p-1)} \\
&\leq \mathbb{1}^\top y^{(p,c,s-1)} + \frac{e^\epsilon - 1}{1 - e^{-\epsilon}}(1 - e^{-\epsilon\xi^{(p,c,s)}\bar{\sigma}^{(p,c,s)}\lambda}) z_c^{(p,s-1)}\hat{\Phi}^{(p-1)} \\
&= \mathbb{1}^\top y^{(p,c,s-1)} + e^\epsilon(z_c^{(p,s-1)} - z_c^{(p,s)})\hat{\Phi}^{(p-1)}.
\end{aligned}$$

Next we add all steps of the while-loop for a customer c. Let c^- denote the previous customer.

$$\begin{aligned}\mathbb{1}^\top y^{(p,c)} &\le \mathbb{1}^\top y^{(p,c^-)} + \sum_{s=1}^{s_c^{(p)}} e^\epsilon (z_c^{(p,s-1)} - z_c^{(p,s)}) \hat{\Phi}^{(p-1)} \\ &= \mathbb{1}^\top y^{(p,c^-)} + e^\epsilon \hat{\Phi}^{(p-1)} (z_c^{(p-1)} - z_c^{(p)}).\end{aligned}$$

We can now bound the total growth within a phase:

$$\begin{aligned}\mathbb{1}^\top y^{(p)} &\le \mathbb{1}^\top y^{(p-1)} + e^\epsilon \hat{\Phi}^{(p-1)} \sum_{c \in \mathcal{A}^{(p-1)}} (z_c^{(p-1)} - z_c^{(p)}) \\ &= \mathbb{1}^\top y^{(p-1)} + e^\epsilon \mathbb{1}^\top y^{(p-1)} \frac{\sum_{c \in \mathcal{A}^{(p-1)}} (z_c^{(p-1)} - z_c^{(p)})}{\sum_{c \in \mathcal{A}^{(p-1)}} z_c^{(p-1)}}.\end{aligned}$$

Finally, by applying $\ln x \le x - 1$ on y and $1 - x \le \ln \frac{1}{x}$ on z we get

$$\ln\left(\frac{\mathbb{1}^\top y^{(p)}}{\mathbb{1}^\top y^{(p-1)}}\right) \le \frac{\mathbb{1}^\top y^{(p)} - \mathbb{1}^\top y^{(p-1)}}{\mathbb{1}^\top y^{(p-1)}} \le e^\epsilon \cdot \ln\left(\frac{\sum_{c \in \mathcal{A}^{(p-1)}} z_c^{(p-1)}}{\sum_{c \in \mathcal{A}^{(p-1)}} z_c^{(p)}}\right).$$

□

Theorem 18 *If the algorithm returns $b_c = \sum_{b \in \mathcal{B}_c} x_{c,b} b$ $(c \in \mathcal{C})$, then*

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))_r \le (1 + 4\epsilon)\sigma\lambda.$$

Moreover $\mathbb{1}^\top y^{(t)} \le e^{2\epsilon\Xi}$, where t is the number of phases.

Proof Using Lemma 17, $\mathcal{A}^{(p-1)} \supseteq \mathcal{A}^{(p)}$, and

$$\sum_{c \in \mathcal{A}^{(t-1)}} z_c^{(t)} \ge \max_{c \in \mathcal{A}^{(t-1)}} z_c^{(t)} \ge \max_{c \in \mathcal{A}^{(t-1)}} z_c^{(t-1)} e^{-\epsilon} \ge e^{-\epsilon(\Xi+1)}$$

we get

$$\begin{aligned}\ln\left(\frac{\mathbb{1}^\top y^{(t)}}{|\mathcal{R}|}\right) &= \ln\left(\frac{\mathbb{1}^\top y^{(t)}}{\mathbb{1}^\top y^{(0)}}\right) = \sum_{p=1}^{t} \ln\left(\frac{\mathbb{1}^\top y^{(p)}}{\mathbb{1}^\top y^{(p-1)}}\right) \le e^\epsilon \sum_{p=1}^{t} \ln\left(\frac{\sum_{c \in \mathcal{A}^{(p-1)}} z_c^{(p-1)}}{\sum_{c \in \mathcal{A}^{(p-1)}} z_c^{(p)}}\right) \\ &\le e^\epsilon \sum_{p=1}^{t-1} \ln\left(\frac{\sum_{c \in \mathcal{A}^{(p-1)}} z_c^{(p-1)}}{\sum_{c \in \mathcal{A}^{(p)}} z_c^{(p)}}\right) + e^\epsilon \cdot \ln\left(\frac{\sum_{c \in \mathcal{A}^{(t-1)}} z_c^{(t-1)}}{\sum_{c \in \mathcal{A}^{(t-1)}} z_c^{(t)}}\right) \\ &= e^\epsilon \ln\left(\frac{\sum_{c \in \mathcal{A}^{(0)}} z_c^{(0)}}{\sum_{c \in \mathcal{A}^{(t-1)}} z_c^{(t)}}\right) \le e^\epsilon \ln\left(|\mathcal{C}| e^{\epsilon(\Xi+1)}\right).\end{aligned}$$

The algorithm ends when $z_c \le e^{-\epsilon\Xi}$ for all $c \in \mathcal{C}$, i.e. when $\min_{c\in\mathcal{C}} \sum_{p=1}^{t} \sum_{s=1}^{s_c^{(p)}} \xi^{(p,c,s)} \bar{\sigma}^{(p,c,s)} \lambda \ge \Xi$. Furthermore, $\bar{\sigma}^{(p,c,s)} \le (1+\epsilon)\sigma$ by definition, and $e^\epsilon \epsilon < 1$ and $(\frac{\epsilon}{3} + e^\epsilon)(1+\epsilon) < 1 + 4\epsilon$ for $0 < \epsilon \le \frac{1}{2}$. We obtain:

$$\begin{aligned}
\ln(\mathbb{1}^\top y^{(t)}) &\le e^\epsilon \ln|\mathcal{C}| + \ln|\mathcal{R}| + e^\epsilon\epsilon(\Xi+1) \\
&\le e^\epsilon \ln|\mathcal{C}| + \ln|\mathcal{R}| + 1 + e^\epsilon\epsilon\Xi = \epsilon(\tfrac{\epsilon}{3} + e^\epsilon)\Xi \\
&\le \epsilon(\tfrac{\epsilon}{3} + e^\epsilon) \min_{c\in\mathcal{C}} \sum_{p=1}^{t} \sum_{s=1}^{s_c^{(p)}} \xi^{(p,c,s)} \bar{\sigma}^{(p,c,s)} \lambda \\
&\le \epsilon(\tfrac{\epsilon}{3} + e^\epsilon)(1+\epsilon)\sigma\lambda \min_{c\in\mathcal{C}} \sum_{p=1}^{t} \sum_{s=1}^{s_c^{(p)}} \xi^{(p,c,s)} \\
&\le \epsilon(1+4\epsilon)\sigma\lambda \min_{c\in\mathcal{C}} X_c^{(t)}.
\end{aligned}$$

Note that as $(\frac{\epsilon}{3} + e^\epsilon) \le 2$ for $0 < \epsilon \le \frac{1}{2}$ the above calculation also shows $\mathbb{1}^\top y^{(t)} \le e^{2\epsilon\Xi}$. Combining the convexity of the resource consumption functions g_c with the above inequality we obtain the global approximation guarantee

$$\begin{aligned}
\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} \left(g_c \left(\sum_{b\in\mathcal{B}_c} x_{c,b} b \right) \right)_r &\le \max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} \sum_{b\in\mathcal{B}_c} x_{c,b} (g_c(b))_r \\
&= \max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} \frac{\sum_{p=1}^{t} \sum_{s=1}^{s_c^{(p)}} \xi^{(p,c,s)} (a^{(p,c,s)})_r}{X_c^{(t)}} \\
&\le \frac{\max_{r\in\mathcal{R}} \sum_{p=1}^{t} \sum_{c\in\mathcal{C}} \sum_{s=1}^{s_c^{(p)}} \xi^{(p,c,s)} (a^{(p,c,s)})_r}{\min_{c\in\mathcal{C}} X_c^{(t)}} \\
&= \frac{\max_{r\in\mathcal{R}} \frac{1}{\epsilon} \ln y_r^{(t)}}{\min_{c\in\mathcal{C}} X_c^{(t)}} \le \frac{\frac{1}{\epsilon} \ln(\mathbb{1}^\top y^{(t)})}{\min_{c\in\mathcal{C}} X_c^{(t)}} \le (1+4\epsilon)\sigma\lambda.
\end{aligned}$$

□

Theorem 19 *The algorithm needs* $O((|\mathcal{C}| + |\mathcal{R}|)\log(|\mathcal{C}| + |\mathcal{R}|)\epsilon^{-2})$ *oracle calls.*

Proof To analyze the number of oracle calls we split them into *successful* calls, i.e. those that lead to an iteration of the while-loop, and *unsuccessful* calls.

For every successful call we know by the definition of ξ that at least one of the resource prices y_r grows by a factor of e^ϵ or the variable z_c shrinks by a factor of e^ϵ. From Theorem 18 we have $1 \le y_r \le \mathbb{1}^\top y \le e^{2\epsilon\Xi}$ for each resource r. Hence the first can happen at most $2|\mathcal{R}|\Xi = O(|\mathcal{R}|(\log|\mathcal{C}| + \log|\mathcal{R}|)\epsilon^{-2})$ times. As $e^{-\epsilon(\Xi+1)} \le z_c \le 1$ for every customer c the second can happen at most $|\mathcal{C}|(\Xi+1) = O(|\mathcal{C}|(\log|\mathcal{C}| + \log|\mathcal{R}|)\epsilon^{-2})$ times. This means that there are $O((|\mathcal{C}| + |\mathcal{R}|)\log(|\mathcal{C}| + |\mathcal{R}|)\epsilon^{-2})$ successful oracle calls.

Whenever a call is unsuccessful we go on to the next customer in $\mathcal{A}$ so these are at most $|\mathcal{C}|$ times the number of phases. In every phase the term $\frac{\mathbb{1}^\top y}{\sum_{c\in\mathcal{A}} z_c}$ grows by at least a factor of $1+\epsilon$ and we know $\frac{|\mathcal{R}|}{|\mathcal{C}|} \le \frac{\mathbb{1}^\top y}{\sum_{c\in\mathcal{A}} z_c} \le \frac{e^{2\epsilon\Xi}}{e^{-\epsilon\Xi}} = e^{3\epsilon\Xi}$. Hence there are $O(\log_{1+\epsilon}(|\mathcal{C}|e^{3\epsilon\Xi})) = O(\frac{\log|\mathcal{C}|+\epsilon\Xi}{\epsilon}) = O(\log(|\mathcal{C}|+|\mathcal{R}|)\epsilon^{-2})$ phases and $O(|\mathcal{C}|\log(|\mathcal{C}|+|\mathcal{R}|)\epsilon^{-2})$ unsuccessful oracle calls. □

Note that the algorithm solves an approximate feasibility version of the MIN–MAX RESOURCE SHARING PROBLEM. Using binary search as described by Young [24] we can use the above algorithm to solve the optimization problem approximately. Again we start with the initial bounds on λ^* provided by Lemma 2 that differ by a multiplicative factor of $|\mathcal{R}|\sigma$. A series of $O(\log\log|\mathcal{R}|)$ calls to the YOUNG–KHANDEKAR ALGORITHM with $\epsilon = \frac{1}{4}$ suffices to reduce this factor to 2σ. Subsequently, we can further tighten these bounds by increasing the precision in a geometric fashion. Suppose we had a lower bound L and an upper bound U with $L \le \lambda^* \le U$ and $U = \sigma\beta L$ with $\beta \ge 1$. Choosing the target value $\lambda := \beta^{\frac{1}{3}} L$ and the error parameter ϵ such that $(1+4\epsilon) = \beta^{\frac{1}{3}}$, we are able to rise the lower bound to $\beta^{\frac{1}{3}} L$ (Lemma 16) or to lower the upper bound to $\sigma\beta^{\frac{2}{3}} L$ (Theorem 18). The new value of β becomes $\beta^{\frac{2}{3}}$ and within $O(\log\frac{1}{\omega})$ iterations we reach $\beta \le 1+\omega$. As the runtime of every step is proportional to ϵ^{-2} and ϵ decreases geometrically, the runtime is dominated by the very last iteration. This leads to the following:

Theorem 20 *Given an instance of the* MIN–MAX RESOURCE SHARING PROBLEM *with σ-optimal block solvers for some given $\sigma \ge 1$. Then a $\sigma(1+\omega)$-approximate solution can be computed in $O(\theta(|\mathcal{C}|+|\mathcal{R}|)\log(|\mathcal{C}|+|\mathcal{R}|)(\log\log|\mathcal{R}|+\omega^{-2}))$ time.*

This generalizes Young's [24] result to the nonlinear case and weak oracles. The same running time bound was claimed by Khandekar [14]. It is only slightly worse than Theorem 8.

8 Application: global routing in chip design

The MIN–MAX RESOURCE SHARING PROBLEM has many applications (e.g. [7,8,13]). Our practical work was motivated by global routing in chip design. Here we are given an undirected graph G with edge capacities $u : E(G) \to \mathbb{R}_+\setminus\{0\}$ and a finite set $\mathcal{N}$ of *nets*. For each net $n \in \mathcal{N}$, we are given a set $\mathcal{D}_n$ of nonempty subsets of $V(G)$, called *pins*. Let $\mathcal{T}_n$ be a set of *Steiner forests* for $\mathcal{D}_n$ in G, i.e., forests T in G with $\bigcup \mathcal{D}_n \subseteq V(T)$ for which contracting each pin results in a tree. In our application, $\mathcal{T}_n$ is the set of all Steiner forests for $\mathcal{D}_n$ in a given subgraph of G (which depends on n).

Furthermore, we have *wire widths* $w : \mathcal{N}\times E(G) \to \mathbb{R}_+\setminus\{0\}$, including the minimum spacing required to neighbouring wires. For the practically relevant objectives, power consumption and manufacturing yield loss, the cost of using an edge $e \in E(G)$ for net $n \in \mathcal{N}$ can be reduced by allocating extra space in addition to the minimally required value $w(n,e)$. We model these spacing-dependent costs by functions $\gamma_{n,e} : \mathbb{R}_+ \to \mathbb{R}_+$ for each $n \in \mathcal{N}$ and $e \in E(G)$. Here $\gamma_{n,e}(s)$ is the estimated contribution to the objective function if e is used by net n with allocated space $w(n,e)+s$.

Both for power and yield optimization, the functions $\gamma_{n,e}$ are convex (they are roughly $s \mapsto c_1 + \frac{c_2}{s}$ for constants $c_1, c_2 > 0$). See [15] and [22] for details.

For wiring length minimization, the traditional objective function in global routing, the functions $\gamma_{n,e}$ are simply constant. This special case was considered previously by Carden, Li and Cheng [4], and by Albrecht [1].

The task is to find a Steiner forest $T_n \in \mathcal{T}_n$ and an extra space assignment $s_n : E(T_n) \to \mathbb{R}_+$ for each $n \in \mathcal{N}$ such that $\sum_{n \in \mathcal{N}: e \in E(T_n)} (w(n,e) + s_n(e)) \leq u(e)$ for each $e \in E(G)$, and the total cost

$$\sum_{n \in \mathcal{N}} \sum_{e \in E(T_n)} \gamma_{n,e}(s_n(e)) \tag{6}$$

is (approximately) minimized.

We replace the objective function by a constraint, imposing an upper bound Γ on the total cost (we can apply binary search to find the optimum value of Γ approximately). However, even deciding whether there exists a feasible solution is *NP*-complete since it contains the edge-disjoint paths problem. Therefore we first consider a fractional relaxation. Let $\chi(T) \in \{0,1\}^{E(G)}$ denote the edge-incidence vector of a Steiner forest T. Instead of an element of

$$\mathcal{B}_n^{\text{int}} := \{(\chi(T), s) \mid T \in \mathcal{T}_n,\ s \in \mathbb{R}_+^{E(G)},\ s_e = 0 \text{ for } e \notin E(T)\}$$

we look for an element of $\mathcal{B}_n := \text{conv}\left(\mathcal{B}_n^{\text{int}}\right)$ for each $n \in \mathcal{N}$. We obtain an instance of the MIN–MAX RESOURCE SHARING PROBLEM by setting $\mathcal{C} := \mathcal{N}$, $\mathcal{R} := E(G) \,\dot\cup\, \{O\}$, where O is an extra resource representing the objective function, and defining resource consumption functions $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$ by

$$\begin{aligned} (g_c(x,s))_e &:= (x_e w(c,e) + s_e)/u(e) \qquad (e \in E(G)) \\ (g_c(x,s))_O &:= \left(\sum_{e \in E(G): x_e > 0} x_e \gamma_{c,e}(s_e/x_e) \right) \Big/ \Gamma \end{aligned} \tag{7}$$

for each $c \in \mathcal{C}$ and $(x,s) \in \mathcal{B}_c$.

We now show how to implement the block solvers. First we prove that the structure of these resource consumption functions allows to optimize over $\mathcal{B}_c^{\text{int}}$ instead of $\mathcal{B}_c$ for each $c \in \mathcal{C}$:

Lemma 21 *Let $c \in \mathcal{C}$ and $y \in \mathbb{R}_+^{\mathcal{R}}$. Then* $\inf_{b \in \mathcal{B}_c^{int}} y^\top g_c(b) = \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$.

Proof Let $b^* = (x^*, s^*) \in \mathcal{B}_c$. We have $b^* = \sum_{j=1}^k \mu_j b^j$ for some $b^1 = (x^1, s^1), \ldots, b^k = (x^k, s^k) \in \mathcal{B}_c^{\text{int}}$, $\mu_1, \ldots, \mu_k > 0$ with $\sum_{j=1}^k \mu_j = 1$, and $k \in \mathbb{N}$.

For $1 \leq j \leq k$, let $\bar{b}^j := (\bar{x}^j, \bar{s}^j) \in \mathcal{B}_c^{\text{int}}$ with $\bar{x}_e^j := x_e^j$ for each $e \in E(G)$ and

$$\bar{s}_e^j := \begin{cases} x_e^j s_e^*/x_e^* & : \ x_e^* > 0 \\ 0 & : \ x_e^* = 0 \end{cases}$$

Then $b^* = \sum_{j=1}^{k} \mu_j \bar{b}^j$. Using the fact that $b \mapsto (g_c(b))_r$ as defined in (7) is linear for all $r \in \mathcal{R}\backslash\{O\}$, and $\bar{s}_e^j / \bar{x}_e^j = s_e^* / x_e^*$ for $1 \le j \le k$ and $e \in E(G)$ with $\bar{x}_e^j > 0$, we get

$$\begin{aligned} y^\top g_c(b^*) &= \sum_{e \in E(G)} y_e \left(g_c \left(\sum_{j=1}^{k} \mu_j \bar{b}^j \right) \right)_e + \frac{y_O}{\Gamma} \left(\sum_{e \in E(G): x_e^* > 0} x_e^* \gamma_{c,e}(s_e^*/x_e^*) \right) \\ &= \sum_{e \in E(G)} y_e \sum_{j=1}^{k} \mu_j (g_c(\bar{b}^j))_e + \frac{y_O}{\Gamma} \left(\sum_{j=1}^{k} \mu_j \sum_{e \in E(G): x_e^j > 0} x_e^j \gamma_{c,e}(\bar{s}_e^j / x_e^j) \right) \\ &= \sum_{j=1}^{k} \mu_j y^\top g_c(\bar{b}^j) \ge \sum_{j=1}^{k} \mu_j \inf_{b \in \mathcal{B}_c^{\text{int}}} y^\top g_c(b) = \inf_{b \in \mathcal{B}_c^{\text{int}}} y^\top g_c(b). \end{aligned}$$

□

Finding a $b^* \in \mathcal{B}_c^{\text{int}}$ for $c \in \mathcal{C}$ with $y^\top g_c(b^*) \le \sigma \inf_{b \in \mathcal{B}_c^{\text{int}}} y^\top g_c(b)$ is equivalent to finding a $T^* \in \mathcal{T}_c$ with $\bar{y}^\top \chi(T^*) \le \sigma \min_{T \in \mathcal{T}_c} \bar{y}^\top \chi(T)$, where $\bar{y} := (\bar{y}_e)_{e \in E(G)}$ is defined by

$$\bar{y}_e := \inf_{s \ge 0} \left(y_e \frac{w(c,e) + s}{u(e)} + y_O \frac{\gamma_{c,e}(s)}{\Gamma} \right) \tag{8}$$

for $e \in E(G)$. Therefore, after adding zero cost edges connecting each pair of vertices belonging to the same pin, an approximation algorithm for the Steiner tree problem in weighted graphs can be used to implement the oracle function f_c.

We can also model timing constraints by defining additional resources similar to O, but involving only a subset of nets each: the delay along a path (sequence of nets) is roughly proportional to the weighted sum of electrical capacitances of the involved nets (see [22]). However, we have not yet implemented this.

Randomized rounding. By solving the above-defined instance of the MIN–MAX RESOURCE SHARING PROBLEM we obtain for each $c \in \mathcal{C}$ an element of $\mathcal{B}_c$, more precisely a convex combination $\sum_{b \in \mathcal{B}_c^{\text{int}}} x_{c,b} b$ of the elements of $\mathcal{B}_c^{\text{int}}$ returned by the block solver f_c. Since we eventually need an element of $\mathcal{B}_c^{\text{int}}$ for each $c \in \mathcal{C}$, it is natural to apply randomized rounding, choosing each $b \in \mathcal{B}_c^{\text{int}}$ with probability $x_{c,b}$, for each $c \in \mathcal{C}$ independently. This actually works very well in our application. One reason is that both presented algorithms compute a solution x for which not only $\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(\sum_{b \in \mathcal{B}_c} x_{c,b}(b))_r$, but even $\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} \sum_{b \in \mathcal{B}_c} x_{c,b}(g_c(b))_r$ is bounded by $\sigma(1+\omega)\lambda^*$ (cf. the proofs of Lemma 3 and Theorem 18). Therefore the following theorem bounds the increase of the maximum resource utilization. This strengthens a well-known result of Raghavan and Thompson [19].

Theorem 22 *Let $\mathcal{C}$ and $\mathcal{B}_c$ ($c \in \mathcal{C}$) be finite sets and $x_{c,b} \ge 0$ for all $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$ with $\sum_{b \in \mathcal{B}_c} x_{c,b} = 1$ for all $c \in \mathcal{C}$. Let $g_c(b) \in \mathbb{R}_+^{\mathcal{R}}$ for $b \in \mathcal{B}_c$ and $c \in \mathcal{C}$, and $\lambda := \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} \sum_{b \in \mathcal{B}_c} x_{c,b}(g_c(b))_r$.*

Consider a "randomly rounded" solution, $\hat{b}_c \in \mathcal{B}_c$ for $c \in \mathcal{C}$, given as follows. Independently for all $c \in \mathcal{C}$ we choose $b \in \mathcal{B}_c$ as $\hat{b}_c$ with probability $x_{c,b}$. Let $\hat{\lambda} := \max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (g_c(\hat{b}_c))_r$.

For $r \in \mathcal{R}$ let $\rho_r := \max\{(g_c(b))_r/\lambda : b \in \mathcal{B}_c, c \in \mathcal{C}, x_{c,b} > 0\}$, and let $\delta > 0$. Then $\hat{\lambda} \le \lambda(1+\delta)$ with probability at least $1 - \sum_{r\in\mathcal{R}} e^{-h(\delta)/\rho_r}$, where $h(\delta) := (1+\delta)\ln(1+\delta) - \delta$.

Proof Consider any resource $r \in \mathcal{R}$. Note that $\frac{(g_c(\hat{b}_c))_r}{\rho_r\lambda}$, $c \in \mathcal{C}$, are independent random variables in $[0, 1]$. The sum of the expectations of these $|\mathcal{C}|$ random variables is at most $\frac{1}{\rho_r}$. Hence, by a lemma due to Raghavan and Spencer (a variant of Chernoff's bound, see Raghavan [18]), their sum is greater than $\frac{1+\delta}{\rho_r}$ with probability less than $e^{-\frac{h(\delta)}{\rho_r}}$. This implies that $\sum_{c\in\mathcal{C}}(g_c(\hat{b}_c))_r > \rho_r\lambda \cdot \frac{1+\delta}{\rho_r} = (1+\delta)\lambda$ with probability less than $e^{-\frac{h(\delta)}{\rho_r}}$. By summing the failure probabilities of all resources we obtain the theorem. □

Since ρ_r is small for most resources in our application, δ can be chosen quite small to have positive success probability. In practice, only few violations occur, i.e. $\sum_{c\in\mathcal{C}}(g_c(\hat{b}_c))_r > 1$ only for a small number of resources $r \in \mathcal{R}$, and these violations can be eliminated easily by postprocessing ("ripup and reroute"). We do not discuss details here since this paper focuses on the MIN–MAX RESOURCE SHARING PROBLEM.

9 Implementation aspects

We discuss in this section some key aspects of our implementation of the (PARALLEL) RESOURCE SHARING ALGORITHM which are important for obtaining good running times in practice, and which are needed to understand the experimental results in Sect. 10.

The graph. We exploit the special structure of global routing graphs. We have a three-dimensional grid with alternating preference directions in the layers and no edges orthogonal to the preference direction. More precisely, the vertex set is $\{0, \ldots, n_x\} \times \{0, \ldots, n_y\} \times \{0, \ldots, n_z\}$, and a vertex pair $\{(x, y, z), (x', y', z')\}$ is an edge if and only if $|x - x'| + |y - y'| + |z - z'| = 1$ and $(x = x' \vee z \not\equiv i \pmod 2)$ and $(y = y' \vee z \equiv i \pmod 2)$. Here $i \in \{0, 1\}$ determines the preference directions of the layers.

Such a graph, and any vertex labels and edge costs, can obviously be stored very efficiently. Adjacency lists are not needed at all.

The oracle. By Lemma 21 and the subsequent remark, the block solvers can be implemented by an (approximation) algorithm for the Steiner tree problem in weighted graphs. Given a connected undirected graph G with edge costs $c : E(G) \to \mathbb{R}_+$ and a terminal set $D \subseteq V(G)$, we look for a Steiner tree for D in G, i.e. a tree T in G with $D \subseteq V(T)$, whose cost $c(T) := \sum_{e\in E(T)} c(e)$ is to be (approximately) minimized. An approximation ratio of 2 is guaranteed by the well-known PATH DECOMPOSITION STEINER TREE ALGORITHM:

PATH DECOMPOSITION STEINER TREE ALGORITHM

Input: A connected undirected graph G with edge costs $c : E(G) \to \mathbb{R}_+$, and a terminal set $D \subseteq V(G)$ with $|D| \geq 2$.
Output: A Steiner tree T for D in G whose cost $c(T)$ is at most twice the optimum.

Set $K := D$ and $F := \emptyset$.
while (K, F) has more than one connected component **do**
  Pick a connected component C of (K, F).
  Find a path P of minimum cost $c(P)$ from $V(C)$ to $K \backslash V(C)$ in G.
  Set $K := K \cup V(P)$ and $F := F \cup E(P)$.
return $T := (K, F)$.

Let $(\bar{G}[D], \bar{c})$ denote the subgraph of the metric closure of (G, c) induced by D, and let M be a minimum spanning tree in $(\bar{G}[D], \bar{c})$. The PATH DECOMPOSITION STEINER TREE ALGORITHM can achieve much shorter cost than $\bar{c}(M)$ by connecting to interior vertices of previously computed paths. For guaranteeing an approximation ratio of 2 we show the folklore result that $\bar{c}(M)$ is never exceeded:

Lemma 23 *The* PATH DECOMPOSITION STEINER TREE ALGORITHM *returns a Steiner tree T for D in G with $c(T) \leq \bar{c}(M)$.*

Proof Let (K_i, F_i) be the forest (K, F) at the end of the ith iteration. We show by induction that there is a set $M_i \subseteq E(M)$ such that $T_i := (K_i, F_i \cup M_i)$ is a tree and $c(F_i) + \bar{c}(M_i) \leq \bar{c}(M)$. This is certainly true for $i = 0$, i.e. after initialization. Now let $i \geq 1$, P_i the path computed in iteration i, P_i' the path connecting the endpoints of P_i in T_{i-1}, and e an edge in P_i' that leaves the connected component C selected in iteration i. Because $e \in E(M_{i-1})$ and P_i is a shortest path from C to any other connected component, we have $\bar{c}(e) \geq c(P_i)$, hence setting $M_i := M_{i-1} \setminus \{e\}$ we get $c(F_i) + \bar{c}(M_i) = c(F_{i-1}) + c(P_i) + \bar{c}(M_{i-1}) - \bar{c}(e) \leq c(F_{i-1}) + \bar{c}(M_{i-1})$. □

This actually proves an approximation ratio of $2 - 2/|D|$. The main subroutine of the PATH DECOMPOSITION STEINER TREE ALGORITHM is Dijkstra's shortest path algorithm, which we implemented with various well-known speed-up techniques.

Lower bounds. As described in Sect. 5, we use lower bounds $l_c \in \mathbb{R}_+^{\mathcal{R}}$ to decide if we can reuse the last solution found for customer $c \in \mathcal{C}$. Because for practically all nets there is a Steiner forest not using a particular given edge, we set $(l_c)_r := 0$ for $r \in \mathcal{R} \backslash \{O\}$, and $(l_c)_O := (g_c(f_c(0, \ldots, 0, 1)))_O / 2$, as $\sigma = 2$ with our implementation of the oracle.

Dynamic adaptation of Γ. Instead of narrowing down the bound Γ on (6) by binary search (thus running the PARALLEL RESOURCE SHARING ALGORITHM several times), it turns out that one obtains very good solutions also by starting with a fairly large value of Γ and dynamically reducing it from phase to phase of the algorithm if the congestion in the extra resource O that models the objective function is not too large.

We first compute a lower bound Γ^{est} on $\sigma\Gamma^*$, where Γ^* is the minimum of (6). To this end, we define g_c as in Sect. 8 for $c \in \mathcal{C}$ with $\Gamma = 1$, compute $x^c := g_c(f_c(y))$

with $y_O := 1$ and $y_e := 0$ for $e \in E(G)$, and set

$$\Gamma^{\text{est}} := \sum_{c \in \mathcal{C}} x_O^c.$$

We then start the PARALLEL RESOURCE SHARING ALGORITHM with $\Gamma := k\Gamma^{\text{est}}$, where $k > 1$ depends on the optimization objective. For wiring length, $k := 1.1$ suffices in practice to obtain a feasible solution, while for power or yield optimization larger values are necessary. If $\lambda_O^{(p)}$ falls below a threshold λ^{thr} at the end of a phase p, we reduce Γ by a factor

$$\frac{\beta\lambda^{\text{thr}} + (1-\beta)\lambda_O^{(p)}}{\lambda_O^{(p)}}$$

with $0 \leq \beta \leq 1$. This is equivalent to scaling g_c for each $c \in \mathcal{C}$. In practice we set $\beta := \frac{\epsilon}{10}$ and $\lambda^{\text{thr}} := 0.95$. The experimental data presented in Sect. 10 show that this approach yields very good results.

Weighting of resource prices. Typically, on large global routing instances with millions of nets, the relative contribution

$$\frac{\gamma_{c,e}(s)}{\Gamma}$$

to the objective value by using edge $e \in E(G)$ in a Steiner forest $T \in \mathcal{T}_c$ for customer $c \in \mathcal{C}$ is below 10^{-8} for all spacing values $s \in \mathbb{R}_+$, while $\frac{w(c,e)+s}{u(e)} \geq 10^{-2}$. Although resource prices grow exponentially, for reasonable values of ϵ the contribution of $y_O \frac{\gamma_{c,e}(s)}{\Gamma}$ to (8) is negligible in the first phases of the RESOURCE SHARING ALGORITHM. Hence, the algorithm *spreads* the wiring very far across the chip area in the first phases, at the cost of increased wiring length. In later phases, when y_O has increased sufficiently, many nets are rerouted with shorter wiring length. We observed considerably faster convergence towards an optimum solution in practice by duplicating the resource O modeling the objective function $\lceil\phi\rceil$ times, where ϕ is chosen such that

$$\phi \sum_{c \in \mathcal{C}} \sum_{e \in E(G)} \frac{\gamma_{c,e}(0)}{\Gamma} = \sum_{c \in \mathcal{C}} \sum_{e \in E(G)} \frac{w(c,e)}{\max\{u(e), w(c,e)\}},$$

Typically, $\phi < |\mathcal{R}|$, hence the worst-case runtime increases only by a small factor. The practical benefit however outweighs this by far.

Zero capacity edges. The graphs in our global routing instances have a regular grid-like structure which can be represented very memory-efficiently. In order to avoid using adjacency lists, we do not delete edges $e \in E(G)$ with capacity $u(e) = 0$.

On some global routing instances, however, there are nets which can be routed only by using one or more edges with zero capacity. Although from a theoretical standpoint, such an instance is infeasible, this often is caused by inaccuracies in the computation of the edge capacities. Hence, in order to generate a reasonable solution

for such instances, we do not count zero capacity edges as resources, but require the oracle to return a Steiner forest using a minimum number of zero capacity edges.

Numerical precision. If numbers are not represented symbolically, but explicitly by listing their (binary) digits, precision of calculations is of course limited. While there are software packages for calculating with arbitrary precision (only limited by memory space), using hardware operations on floating point numbers as defined in the IEEE 754 standard [11] is much faster. Our implementation uses 64-bit double-precision IEEE-754 floating point numbers, which are supported in hardware on all current microprocessors. Switching from double-precision to extended-precision numbers (with 80 bits instead of 64), also directly supported in hardware, already causes a significant runtime penalty, but offers almost no benefit in practice.

With double-precision IEEE-754 floating point numbers $a, b \in \mathbb{Q}_+ \setminus \{0\}$, the condition that

$$a + b > \max\{a, b\} \tag{9}$$

holds only if $\max\{a, b\}/\min\{a, b\} < 2^{52}$, as the mantissa of double-precision numbers encompasses 52 bits (11 bits are used for the exponent, and one bit for the sign). Ensuring this condition is important because otherwise resources can be used at an effective cost of zero, causing unnecessary routing detours.

Although resources are introduced only for edges with positive capacity, the oracle operates directly on the global routing graph G. Because we want to minimize the number of zero capacity edges used, we have to impose costs on zero capacity edges which are greater than the sum of costs for using edges with positive capacity. Further, the oracle has to increment total costs a of partial routings for a net by costs b for adding an edge, so

$$(1 - \xi)|E(G)| \max\{1, (\xi|E(G)| - 1)x\} < 2^{52}$$

is necessary and sufficient to guarantee (9) if $\xi > 0$ is the fraction of edges with positive capacity, and x is the cost ratio between the most expensive and the cheapest among them.

The amount consumed of the corresponding resources differs by a factor of at most

$$k_1 := \max_{e_1, e_2 \in E^+} \frac{w(c, e_1)u(e_2)}{w(c, e_2)u(e_1)},$$

for a customer $c \in \mathcal{C}$, where $E^+ := \{e \in E(G) : u(e) > 0\}$, so we can guarantee (9) if

$$\frac{\max_{r \in \mathcal{R} \setminus \{O\}} e^{\epsilon \alpha_r}}{\min_{r \in \mathcal{R} \setminus \{O\}} e^{\epsilon \alpha_r}} < \frac{2^{52}}{\frac{1}{4}|E(G)|^2 k_1} \tag{10}$$

and

$$k_2 := \max_{e_1, e_2 \in E^+, s_1, s_2 \in \mathbb{R}_+} \frac{\gamma_{c,e_1}(s_1)}{\gamma_{c,e_2}(s_2)} < \frac{2^{52}}{\frac{1}{4}|E(G)|^2}. \tag{11}$$

Usually (10) implies (11) because $k_2 \leq k_1$. However, we cannot ensure (10) on the largest of today's global routing instances because the right-hand side is too small (it is approximately 1). In practice however it suffices to ensure that costs for using an edge with zero capacity are 100 times higher than the highest cost for using an edge with positive capacity; we observed only few exceptions in which this did not minimize the number of zero capacity edges in a shortest path to be found in a call to the block solver. Moreover, not more than ten zero capacity edges are usually needed per path (in most cases only for accessing "blocked" pins). Therefore it is relatively safe in practice to replace the factor $|E(G)|^2$ in (10) by 2^{10}, providing a ratio of $2^{44}/k_1$ between maximum and minimum edge congestion costs. We can assume $k_1 \leq 128 = 2^7$ and replace the costs $e^{\epsilon\alpha_r}$ for $r \in \mathcal{R}\backslash\{O\}$ by

$$e^{\epsilon \max\{\min\{\alpha_r, p\lambda^p_{\max}\}, p\lambda^p_{\min}\}} \tag{12}$$

in phase p of the RESOURCE SHARING ALGORITHM, with $\lambda^p_{\min} = \lambda^p_{\max} - \frac{\ln 2^{37}}{\epsilon p}$. For example, with $\epsilon = 1$, the interval $[\lambda^p_{\min}, \lambda^p_{\max}]$ can be set to $[0, 25.6]$ in the first phase, $[0.744, 1.256]$ in the 50th phase and $[0.872, 1.128]$ in the 100th phase of the RESOURCE SHARING ALGORITHM. This suffices in practice because routable chips typically have $\lambda^* \in [0.9, 1]$, and on unroutable chips the value of λ^* is not of high interest if $\lambda^* > 1.1$.

As expected, if the maximum congestion exceeds $\lambda^p_{\max}$ at some time during phase p of the algorithm ($1 \leq p \leq t$), it often increases significantly from that point. Bounding edge resource costs from below by $e^{\epsilon p \lambda^p_{\min}}$ in (12) with $\lambda^p_{\min} < \lambda^*$ does not cause significantly different behaviour of the algorithm.

10 Experimental results

In this section we show experimental results of the PARALLEL RESOURCE SHARING ALGORITHM on global routing instances which originate from recent industrial chips. We also compare the two algorithms that we presented.

Table 2 shows results for wiring length minimization with different choices of ϵ and t. In our instances we always have $\lambda^* > \frac{1}{2}$, and on instances with $\lambda^* > 1$ we are interested only in a certificate of infeasibility. Hence we do not perform scaling as in Theorems 8–10, but just run the core algorithm. Moreover, in these experiments we dynamically adapt the bound Γ on the total objective value as discussed in Sect. 9.

We choose the number t of phases such that $t\epsilon = 125$, which turns out to be sufficient in practice, and set the parameter τ for reusing previously computed solutions, and defining the acceptance criterion in the procedure PARALLELALLOCATERESOURCES, to $1 + 0.05\epsilon$. We set the initial bound $\Gamma := 1.1\Gamma^{\text{est}}$.

In Table 2, λ_{edges} is the value of $\max_{r\in\mathcal{R}\backslash\{O\}} \sum_{c\in\mathcal{C}} \left(g_c \left(\sum_{b\in\mathcal{B}_c} x_{c,b} b\right)\right)_r$ at the end of the PARALLEL RESOURCE SHARING ALGORITHM. Note that in many cases λ_O and λ_{edges} are close to 0.95. This is due to the dynamic scaling of Γ discussed in Sect. 9, using a threshold value $\lambda^{\text{thr}} := 0.95$.

Table 2 Comparison of different choices for ϵ, optimizing wiring length

Chip	ϵ	Phases	λ_O	λ_{edges}	λ_{dual}	Obj. / Γ^{est} ratio	Running time	Speedup
Nouzada	0.100	1250	0.9499	0.9652	0.9488	1.014	0:04:48	
$\lvert\mathcal{C}\rvert = 126{,}365$	0.250	500	0.9499	0.9653	0.9485	1.014	0:01:50	
$\lvert\mathcal{R}\rvert = 76{,}543$	1.000	125	0.9499	0.9680	0.9419	1.018	0:00:19	4.34
Christopher	0.100	1250	0.9499	0.9630	0.9486	1.021	0:04:55	
$\lvert\mathcal{C}\rvert = 96{,}883$	0.250	500	0.9499	0.9630	0.9480	1.021	0:01:54	
$\lvert\mathcal{R}\rvert = 97{,}124$	1.000	125	0.9499	0.9640	0.9438	1.023	0:00:20	5.48
Omnea	0.100	1250	0.9498	0.9630	0.9478	1.023	0:11:02	
$\lvert\mathcal{C}\rvert = 221{,}795$	0.250	500	0.9498	0.9626	0.9479	1.024	0:04:21	
$\lvert\mathcal{R}\rvert = 70{,}227$	1.000	125	0.9499	0.9611	0.9434	1.025	0:00:52	6.18
Estelle	0.100	1250	0.9499	0.9620	0.9474	1.041	0:19:28	
$\lvert\mathcal{C}\rvert = 371{,}733$	0.250	500	0.9498	0.9618	0.9478	1.041	0:07:43	
$\lvert\mathcal{R}\rvert = 168{,}651$	1.000	125	0.9499	0.9605	0.9435	1.043	0:01:42	6.42
Gerhard	0.100	1250	0.9499	0.9569	0.9488	1.022	0:15:59	
$\lvert\mathcal{C}\rvert = 347{,}552$	0.250	500	0.9499	0.9562	0.9486	1.022	0:06:04	
$\lvert\mathcal{R}\rvert = 591{,}579$	1.000	125	0.9499	0.9546	0.9438	1.024	0:01:08	6.04
Etienne	0.100	1250	0.9608	1.0000	0.9569	1.055	0:35:34	
$\lvert\mathcal{C}\rvert = 313{,}718$	0.250	500	0.9612	1.0000	0.9563	1.055	0:14:17	
$\lvert\mathcal{R}\rvert = 244{,}431$	1.000	125	0.9624	1.0000	0.9505	1.058	0:03:16	6.78
Henrik	0.100	1250	0.9499	0.9740	0.9488	1.011	0:23:23	
$\lvert\mathcal{C}\rvert = 440{,}915$	0.250	500	0.9499	0.9734	0.9484	1.011	0:08:58	
$\lvert\mathcal{R}\rvert = 2{,}480{,}869$	1.000	125	0.9499	0.9722	0.9449	1.014	0:01:47	6.42
Emilia	0.100	1250	0.9499	0.9640	0.9500	1.006	0:29:59	
$\lvert\mathcal{C}\rvert = 430{,}273$	0.250	500	0.9499	0.9640	0.9494	1.006	0:11:07	
$\lvert\mathcal{R}\rvert = 7{,}947{,}102$	1.000	125	0.9500	0.9632	0.9435	1.011	0:01:47	5.41
Milena	0.100	1250	0.9499	0.9742	0.9482	1.034	1:12:08	
$\lvert\mathcal{C}\rvert = 1{,}312{,}592$	0.250	500	0.9499	0.9742	0.9481	1.034	0:28:03	
$\lvert\mathcal{R}\rvert = 1{,}659{,}036$	1.000	125	0.9499	0.9717	0.9431	1.037	0:06:05	6.91
Simona	0.100	1250	0.9500	1.0490	0.9848	1.031	1:28:00	
$\lvert\mathcal{C}\rvert = 1{,}406{,}678$	0.250	500	0.9500	1.0500	0.9849	1.031	0:34:18	
$\lvert\mathcal{R}\rvert = 1{,}003{,}448$	1.000	125	0.9500	1.0519	1.0117	1.034	0:06:54	6.19
Guido	0.100	1250	0.9499	0.9646	0.9491	1.017	1:26:04	
$\lvert\mathcal{C}\rvert = 1{,}796{,}234$	0.250	500	0.9499	0.9642	0.9488	1.017	0:33:43	
$\lvert\mathcal{R}\rvert = 2{,}469{,}158$	1.000	125	0.9499	0.9644	0.9446	1.020	0:06:17	6.49
Dirk	0.100	1250	0.9815	3.1982	1.0800	1.069	2:44:40	
$\lvert\mathcal{C}\rvert = 1{,}600{,}898$	0.250	500	0.9838	3.3178	1.0672	1.073	1:05:05	
$\lvert\mathcal{R}\rvert = 3{,}316{,}786$	1.000	125	0.9922	3.7036	1.0127	1.087	0:14:44	6.91
Thilo	0.100	1250	0.9500	0.9790	0.9476	1.042	4:12:09	
$\lvert\mathcal{C}\rvert = 2{,}772{,}390$	0.250	500	0.9500	0.9790	0.9471	1.042	1:40:18	
$\lvert\mathcal{R}\rvert = 2{,}234{,}909$	1.000	125	0.9504	0.9791	0.9403	1.045	0:23:18	7.05

Table 2 continued

Chip	ϵ	Phases	λ_O	λ_{edges}	λ_{dual}	Obj. / Γ^{est} ratio	Running time	Speedup
Martina	0.100	1250	0.9499	0.9790	0.9488	1.026	4:23:05	
$\lvert\mathcal{C}\rvert = 3\,783\,704$	0.250	500	0.9499	0.9791	0.9482	1.026	1:45:10	
$\lvert\mathcal{R}\rvert = 4\,207\,761$	1.000	125	0.9499	0.9790	0.9433	1.029	0:20:14	6.87

All runs used 8 threads in parallel

The value λ_{dual} is

$$\frac{\sum_{c\in\mathcal{C}}(y^{(t)})^\top g_c(f_c(y^{(t)}))}{(y^{(t)})^\top \mathbb{1}}.$$

Hence by Lemma 1 applied to $y^{(t)}$, $\lambda_{\text{dual}}/\sigma$ is a lower bound on λ^*. As shown in Sect. 9, we have $\sigma = 2$ for our implementation of the oracle. In practice however

$$\sigma_{\text{avg}} := \frac{\sum_{c\in\mathcal{C}}(y^{(t)})^\top g_c(f_c(y^{(t)}))}{\sum_{c\in\mathcal{C}} \text{opt}_c(y^{(t)})}$$

is considerably smaller than 2, as many nets have only two pins (in this case an optimum solution is a shortest path), and the PATH DECOMPOSITION STEINER TREE ALGORITHM also returns a solution within a few percent of optimum for most other instances in practice. Of course if $\lambda_{\text{dual}}/\sigma_{\text{avg}} > 1$, this gives an infeasibility proof. On the other hand, if $\lambda_{\text{dual}} > 1$ but $\lambda_{\text{dual}}/\sigma_{\text{avg}} \leq 1$, this gives an infeasibility proof *relative to* $\sigma_{\text{avg}}/\lambda_{\text{dual}}$, i.e. showing that by using block solvers which return only $\sigma_{\text{avg}}/\lambda_{\text{dual}}$-optimal solutions, a feasible solution to the MIN–MAX RESOURCE SHARING PROBLEM cannot be found.

We are therefore almost sure that Dirk is infeasible (although we cannot formally prove this with our oracle). We do not know whether Simona is feasible. For all other instances we found feasible solutions.

The column denoted "obj. / Γ^{est}" shows the ratio between the achieved wiring length and the target value Γ^{est}. The experiments were made on a 3.33 GHz Intel Xeon machine with 144 GB of memory and used 8 processors in parallel. Running times of the PARALLEL RESOURCE SHARING ALGORITHM are given in hh:mm:ss. Speedup over the sequential RESOURCE SHARING ALGORITHM is shown only for $\epsilon = 1.0$ because of the higher running times for smaller values of ϵ. The sequential runs were not done with the PARALLEL RESOURCE SHARING ALGORITHM running on a single processor, but with the sequential RESOURCE SHARING ALGORITHM in order to avoid overhead that is necessary only for the parallelization.

Interestingly, choosing ϵ smaller and t larger does not seem to help much in practice. We get excellent results already for $\epsilon = 1$ and $t = 125$ within less than half an hour even for the largest instances.

For the following experiments we considered only the twelve chips for which we found feasible solutions. Table 3 compares results for minimizing wiring length, critical area (a measure for the manufacturing yield loss; cf. [15]), and power consumption. All runs in this table are made with $\epsilon := 1$, and we set the initial bound $\Gamma := 1.1\,\Gamma^{\text{est}}$ for wiring length optimization, $\Gamma := 1.5\,\Gamma^{\text{est}}$ for power optimization and $\Gamma := 2\,\Gamma^{\text{est}}$ for yield optimization. As above, the column denoted "obj. / Γ^{est} ratio" shows the ratio between the achieved objective value and Γ^{est}. In addition, for each chip and each objective, we determined a lower bound Γ^{lb} on the optimum objective value Γ^* that can be achieved with our implementation of the oracle. This was done by binary search on Γ^*, increasing Γ^{lb} whenever for the current value infeasibility relative to $\sigma_{\text{avg}}/\lambda_{\text{dual}}$ is proven. In this binary search, the resource consumption functions g_c for $c \in \mathcal{C}$ are not scaled during an execution of the core algorithm.

Columns 5–7 evaluate the achieved result w.r.t. all three metrics and show the relative value compared to the run actually optimizing the corresponding objective. The last column shows the running times.

Traditional global routers just try to minimize wiring length (and are often purely heuristic, without any performance guarantee). Table 3 shows that a global routing with almost optimum wiring length is not very good with respect to power consumption and yield. With our algorithm, these objectives can be optimized directly for the first time, in very reasonable running time.

In Table 4 we finally compare our RESOURCE SHARING ALGORITHM with the YOUNG–KHANDEKAR ALGORITHM as presented in Sect. 7. These runs were all made on a 2.93 GHz Intel Xeon machine with 96 GB of memory and used only one processor because we did not parallelize our implementation of the YOUNG–KHANDEKAR ALGORITHM.

In each of these experiments, we have chosen Γ slightly larger than the objective values achieved in the above experiments, hence a feasible solution exists but is not easy to find. The choice of Γ is shown in column 3, and ϵ is set to 1. Columns 4 and 5 show the results of our RESOURCE SHARING ALGORITHM running with $\tau := 1.05$ for reusing previously computed solutions. The YOUNG–KHANDEKAR ALGORITHM requires input parameters λ and σ. We set $\lambda := 1$ because λ^* is slightly smaller than 1, but as it is not clear how to set σ, we ran the algorithm for $\sigma := 1.05$ (columns 7–9) and $\sigma := 2$ (columns 10–12).

For each run, the values λ_O and λ_{edges} for the phase which minimizes $\max\{\lambda_O, \lambda_{\text{edges}}\}$ is shown (this is not necessarily the last phase). The results show that our RESOURCE SHARING ALGORITHM consistently achieves a better congestion with a significantly lower running time. Interestingly, on some of the chips, increasing σ from 1.05 to 2 improves not only running time, but also congestion.

11 Conclusion

We showed that the general MIN–MAX RESOURCE SHARING PROBLEM can be solved efficiently in theory and practice. We improved the best known worst-case running times, applied the algorithm to global routing in chip design, and solved very large practical instances almost optimally.

Table 3 Comparison of different optimization objectives

Chip	Optimization objective	Obj. / Γ^{est} ratio	Obj. / Γ^{lb} ratio	Relative obj. value			Running time
				Wiring length	Critical area	Power	
Nouzada	Wiring length	1.0179	1.0070	1.000	1.088	1.109	0:00:19
	Critical area	1.0239	1.0066	1.043	1.000	1.068	0:00:40
	Power	1.1197	1.0257	1.169	1.197	1.000	0:01:23
Christopher	Wiring length	1.0235	1.0066	1.000	1.116	1.136	0:00:20
	Critical area	1.0278	1.0064	1.053	1.000	1.068	0:00:38
	Power	1.1291	1.0229	1.196	1.216	1.000	0:01:47
Omnea	Wiring length	1.0254	1.0079	1.000	1.044	1.068	0:00:52
	Critical area	1.0396	1.0092	1.023	1.000	1.045	0:01:36
	Power	1.2049	1.0234	1.176	1.202	1.000	0:02:34
Estelle	Wiring length	1.0430	1.0107	1.000	1.066	1.097	0:01:42
	Critical area	1.0491	1.0109	1.037	1.000	1.056	0:02:34
	Power	1.1795	1.0265	1.217	1.233	1.000	0:04:15
Gerhard	Wiring length	1.0243	1.0074	1.000	1.169	1.180	0:01:08
	Critical area	1.0380	1.0086	1.088	1.000	1.113	0:03:27
	Power	1.0929	1.0236	1.288	1.330	1.000	0:04:34
Etienne	Wiring length	1.0577	1.0168	1.000	1.091	1.077	0:03:16
	Critical area	1.1293	1.0277	1.054	1.000	1.060	0:07:08
	Power	1.2347	1.0363	1.201	1.239	1.000	0:07:07
Henrik	Wiring length	1.0138	1.0064	1.000	1.052	1.055	0:01:47
	Critical area	1.0268	1.0100	1.030	1.000	1.043	0:04:47
	Power	1.0654	1.0300	1.146	1.172	1.000	0:07:36
Emilia	Wiring length	1.0112	1.0061	1.000	1.779	1.115	0:01:47
	Critical area	1.5454	1.0540	1.305	1.000	1.460	1:54:26
	Power	1.0489	1.0179	1.169	1.722	1.000	0:18:06
Milena	Wiring length	1.0365	1.0112	1.000	1.103	1.136	0:06:05
	Critical area	1.0584	1.0134	1.074	1.000	1.087	0:14:10
	Power	1.1460	1.0298	1.285	1.301	1.000	0:18:08
Guido	Wiring length	1.0203	1.0072	1.000	1.138	1.115	0:06:17
	Critical area	1.0660	1.0149	1.071	1.000	1.100	0:26:28
	Power	1.1193	1.0287	1.220	1.282	1.000	0:30:39
Thilo	Wiring length	1.0454	1.0158	1.000	1.110	1.098	0:23:18
	Critical area	1.1154	1.0237	1.055	1.000	1.076	0:50:50
	Power	1.1937	1.0322	1.241	1.292	1.000	0:54:39
Martina	Wiring length	1.0288	1.0097	1.000	1.151	1.134	0:20:14
	Critical area	1.0775	1.0166	1.072	1.000	1.101	1:03:32
	Power	1.1298	1.0278	1.232	1.297	1.000	1:10:18

All runs were made with $\epsilon := 1$ using 8 threads in parallel

Table 4 Comparison of our RESOURCE SHARING ALGORITHM and the presented YOUNG–KHANDEKAR ALGORITHM with $\sigma := 1.05$ and $\sigma := 2$

Chip	Optimization objective	Γ/Γ^{est}	Our algorithm			YOUNG–KHANDEKAR ($\sigma = 1.05$)			YOUNG–KHANDEKAR ($\sigma = 2$)		
			λ_O	λ_{edges}	Running time	λ_O	λ_{edges}	Running time	λ_O	λ_{edges}	Running time
Nouzada	Wiring length	1.02	0.9963	1.0013	0:01:39	0.9953	1.0083	0:12:40	0.9963	1.0125	0:08:25
	Critical area	1.03	0.9964	1.0001	0:04:43	0.9933	1.0139	0:22:37	0.9945	1.0062	0:14:45
	Power	1.12	0.9939	0.9968	0:11:48	0.9941	1.0114	0:27:54	0.9963	1.0133	0:18:02
Christopher	Wiring length	1.03	0.9920	1.0046	0:02:17	0.9927	1.0135	0:13:21	0.9936	1.0196	0:08:44
	Critical area	1.03	0.9985	1.0056	0:04:17	0.9966	1.0234	0:21:23	0.9980	1.0193	0:13:26
	Power	1.13	0.9932	0.9994	0:15:23	0.9930	1.0105	0:37:35	0.9954	1.0132	0:24:26
Omnea	Wiring length	1.03	0.9925	1.0030	0:06:31	0.9932	1.0130	0:28:16	0.9944	1.0121	0:18:44
	Critical area	1.04	0.9971	1.0055	0:13:03	0.9969	1.0164	0:42:51	0.9981	1.0196	0:28:01
	Power	1.21	0.9925	1.0033	0:23:42	0.9927	1.0092	0:54:15	1.0007	1.0436	0:35:25
Estelle	Wiring length	1.05	0.9892	0.9989	0:15:00	0.9901	1.0505	0:46:35	0.9913	1.1632	0:30:35
	Critical area	1.05	0.9954	1.0031	0:21:32	0.9949	1.0195	1:07:09	0.9961	1.0238	0:43:52
	Power	1.18	0.9941	1.0020	0:41:01	0.9934	1.0603	1:32:57	1.0012	1.0725	1:01:44
Gerhard	Wiring length	1.03	0.9926	0.9972	0:08:30	0.9936	1.0276	0:46:09	0.9947	1.0156	0:30:18
	Critical area	1.04	0.9989	0.9953	0:30:00	0.9970	1.0450	2:16:07	0.9988	1.0158	1:29:03
	Power	1.10	0.9887	0.9910	0:42:21	0.9874	1.0305	2:01:33	0.9894	1.0129	1:19:32
Etienne	Wiring length	1.06	0.9924	1.0207	0:25:57	0.9920	1.0484	1:22:07	0.9933	1.0489	0:57:17
	Critical area	1.13	0.9907	1.0148	1:01:45	0.9882	1.0481	2:50:50	0.9903	1.0355	2:00:23
	Power	1.24	0.9890	1.0270	1:06:24	0.9872	1.0365	2:36:57	0.9893	1.0443	1:48:25
Henrik	Wiring length	1.02	0.9920	1.0130	0:13:13	0.9969	1.1220	1:45:32	0.9936	1.0430	1:10:54
	Critical area	1.03	1.0059	1.0252	0:36:57	0.9947	1.0596	3:26:17	0.9977	1.0625	2:15:54
	Power	1.07	0.9902	1.0091	1:05:40	0.9889	1.0770	3:55:36	0.9928	1.0336	2:36:06

Table 4 continued

Chip	Optimization objective	$\Gamma/\Gamma^{\text{est}}$	Our algorithm			Young–Khandekar ($\sigma = 1.05$)			Young–Khandekar ($\sigma = 2$)		
			λ_O	λ_{edges}	Running time	λ_O	λ_{edges}	Running time	λ_O	λ_{edges}	Running time
Emilia	Wiring length	1.02	0.9905	1.0032	0:11:42	0.9907	1.0599	3:55:21	0.9918	1.0399	2:32:41
	Critical area	1.55	0.9877	0.9939	16:01:06	0.9785	1.0866	39:14:11	0.9821	1.0301	26:34:58
	Power	1.05	0.9960	0.9936	2:23:26	0.9911	1.0406	12:06:07	0.9927	1.0315	7:58:53
Milena	Wiring length	1.04	0.9925	1.0061	0:53:10	0.9935	1.0642	3:26:55	0.9958	1.1408	2:17:00
	Critical area	1.06	0.9960	1.0072	2:05:35	0.9941	1.0773	8:05:04	0.9957	1.0850	5:21:30
	Power	1.15	0.9912	1.0036	2:50:42	0.9895	1.0729	8:25:11	0.9945	1.1029	5:28:51
Guido	Wiring length	1.03	0.9887	1.0009	0:50:03	0.9887	1.0808	5:13:30	0.9900	1.0249	3:53:45
	Critical area	1.07	0.9947	1.0013	3:58:30	0.9939	1.0647	16:15:05	0.9936	1.0176	10:38:32
	Power	1.12	0.9934	1.0042	4:44:44	0.9916	1.0849	14:42:42	0.9942	1.0223	9:48:18
Thilo	Wiring length	1.05	0.9900	1.0153	2:48:32	0.9900	1.0435	10:53:03	0.9913	1.0366	7:12:47
	Critical area	1.12	0.9895	1.0131	7:32:44	0.9874	1.0676	24:45:37	0.9888	1.0411	16:29:36
	Power	1.20	0.9880	1.0152	8:30:06	0.9862	1.0597	24:35:48	0.9883	1.0529	16:13:21
Martina	Wiring length	1.03	0.9954	1.0230	2:27:38	0.9954	1.0654	14:04:56	0.9963	1.0415	9:04:16
	Critical area	1.08	0.9949	1.0194	9:16:02	0.9920	1.0763	37:22:49	0.9939	1.0396	23:59:45
	Power	1.13	0.9930	1.0200	10:32:09	0.9910	1.0711	33:38:35	0.9933	1.0366	21:52:47

All runs were performed sequentially

Appendix: Comments on Khandekar's Thesis

In Theorem 6.3.1 of his thesis [14] Khandekar develops an algorithm for the feasibility version of the MIN–MAX RESOURCE SHARING PROBLEM which he calls BLOCK-ANGULAR PACKING PROBLEM. This can be used to solve the optimization version in $O(\theta(|\mathcal{R}|+|\mathcal{C}|)\log(|\mathcal{R}|+|\mathcal{C}|)(\log\log|\mathcal{R}|+\omega^{-2}))$ time. Unfortunately his analysis contains two errors. He reduces the BLOCK- ANGULAR PACKING PROBLEM to the SUBSTITUTE MIXED PROBLEM, which he describes in Sect. 6.2.

The first problem arises in Algorithm 6.2 (on page 57) in the update of the z_j-variables. As soon as they fall below the $e^{-\epsilon\Gamma}$-threshold they are not updated any more and from then on the definition of the weights in line (5) of Algorithm 6.1 allows for bigger values. Khandekar's proof of correctness for the SUBSTITUTE MIXED ALGORITHM heavily relies on inequality (6.8). The SUBSTITUTE MIXED PROBLEM is itself reduced to the MIXED PROBLEM, whose analysis is described in Sect. 5.1.3. Since inequality (6.8) is reduced to inequality (5.6), we have to go into details there. The main ingredients for the proof of (5.6) are the inequalities (5.2) and (5.3). He derives them from Corollaries 2.1.2 and 2.1.3, respectively. While the first one is correct, the second one does not hold in the given case. In the middle of the proof of Theorem 2.1.1, which is the foundation of the two corollaries, Khandekar shows (on page 12)

$$(e^{\epsilon}\mathcal{P}_i^r - e^{-\epsilon}\mathcal{L}_i^r)\cdot y_i^{r-1} \geq \frac{y_i^r - y_i^{r-1}}{\epsilon}, \tag{13}$$

where $y_i^r (= e^{\epsilon x_i^r})$ stands for the ith dual variable (named y or z in Chapter 5) after the rth iteration, and $\mathcal{L}_i^r, \mathcal{P}_i^r \geq 0$ denote the change in x_i^r; so we have $y_i^r = e^{\epsilon(\mathcal{P}_i^r - \mathcal{L}_i^r)}\cdot y_i^{r-1}$. He uses $\mathcal{L}_i^r = 0$ in case of the first corollary and $\mathcal{P}_i^r = 0$ in case of the second. Inequality (13) holds as long as the dual variables change according to this definition, but the problem arises in the particular case when the dual variables are not updated. We have $\mathcal{P}_i^r = 0$, $\mathcal{L}_i^r > 0$, and $y_i^r = y_i^{r-1}$. Hence the left-hand side of (13) is negative while the right-hand side is zero. We see that Corollary 2.1.3 can not be used for dual variables that do not change according to the definition.

The second problem arises in Section 6.2.2 where Khandekar applies Fleischer's [6] idea to reduce the number of oracle calls. From page 59 on he uses e^{ϵ}-approximate oracles. He calls an oracle call "fruitful" and hence uses the output x for the computation if $\frac{y^\top f^c(x)/\mathbb{1}^\top y}{z^\top g^c(x)/\mathbb{1}^\top z} \leq e^{\epsilon}$. Otherwise it is unfruitful and the next customer's oracle is queried. In the proof of Lemma 6.2.4 he wants to bound the number of unfruitful oracle calls by showing that every t unfruitful calls the value of the expression $\frac{\mathbb{1}^\top y}{\mathbb{1}^\top z}$ rises by at least a factor of e^{ϵ}. Since $\frac{\mathbb{1}^\top y}{\mathbb{1}^\top z}$ is bounded, this would imply that the total number of such oracle calls can be bounded. Therefore he claims (on page 61) that after an unfruitful oracle call we have

$$\min_{x'\in\mathcal{B}_c} \frac{y^\top f^c(x')/\mathbb{1}^\top y}{z^\top g^c(x')/\mathbb{1}^\top z} > e^{\epsilon}.$$

However, by the definition of e^ϵ-approximate oracles and of unfruitful calls, the output x just satisfies

$$e^\epsilon \min_{x' \in \mathcal{B}_c} \frac{y^\top f^c(x')/\mathbb{1}^\top y}{z^\top g^c(x')/\mathbb{1}^\top z} \geq \frac{y^\top f^c(x)/\mathbb{1}^\top y}{z^\top g^c(x)/\mathbb{1}^\top z} > e^\epsilon$$

and hence $\min_{x' \in \mathcal{B}_c} \frac{y^\top f^c(x')/\mathbb{1}^\top y}{z^\top g^c(x')/\mathbb{1}^\top z} > 1$, which is considerably weaker and not enough to deduce his assertion.

References

1. Albrecht, C.: Global routing by new approximation algorithms for multicommodity flow. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **20**, 622–632 (2001)
2. Bienstock, D.: Potential function methods for approximately solving linear programming problems: theory and practice. Kluwer Academic Publishers, Norwell (2002)
3. Bienstock, D., Iyengar, G.: Approximating fractional packings and coverings in $O(\frac{1}{\epsilon})$ iterations. SIAM J. Comput. **35**, 825–854 (2006)
4. Carden, R.C. IV., Li, J., Cheng, C.-K.: A global router with a theoretical bound on the optimum solution. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **15**, 208–216 (1996)
5. Charikar, M., Chekuri, C., Goel, A., Guha, S., and Plotkin, S.: Approximating a finite metric by a small number of tree metrics. In: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 379–388 (1998)
6. Fleischer, L.K.: Approximating fractional multicommodity flow independent of the number of commodities. SIAM J. Discret. Math. **13**, 505–520 (2000)
7. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. SIAM J. Comput. **37**, 630–652 (2007)
8. Grigoriadis, M.D., Khachiyan, L.D.: Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM J. Optim. **4**, 86–107 (1994)
9. Grigoriadis, M.D., Khachiyan, L.D.: Coordination complexity of parallel price-directive decomposition. Math. Oper. Res. **21**, 321–340 (1996)
10. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 4th edn. Morgan Kaufmann, San Francisco (2007)
11. IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754–2008, IEEE (2008)
12. Intel®64 and IA-32 Architectures Software Developer's Manual, vol. 3A, rev. 037, order number 253668-037US, January 2011. http://www.intel.com
13. Jansen, K., Zhang, H.: Approximation algorithms for general packing problems and their application to the multicast congestion problem. Math. Program. A **114**, 183–206 (2008)
14. Khandekar, R.: Lagrangian relaxation based algorithms for convex programming problems. PhD thesis, Indian Institute of Technology, Delhi (2004)
15. Müller, D.: Optimizing yield in global routing. In: Proceedings of the IEEE International Conference on Computer-Aided Design, 480–486 (2006)
16. Owens, S., Sarkar, S., Sewell, P.: A Better x86 Memory Model: x86-TSO. In: Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics; LNCS 5674, pp. 391–407. Springer, Berlin (2009)
17. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. Math. Oper. Res. **2**, 257–301 (1995)
18. Raghavan, P.: Probabilistic construction of deterministic algorithms: approximating packing integer programs. J. Comput. Syst. Sci. **37**, 130–143 (1988)
19. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**, 365–374 (1987)
20. Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM **37**, 318–334 (1990)
21. Villavicencio, J., Grigoriadis, M.D.: Approximate structured optimization by cyclic block-coordinate descent. In: Fischer, H., Riedmüller, B., Schäffler, S. (eds.) Applied Mathematics and Parallel Computing, pp. 359–371. Physica-Verlag, Heidelberg (1996)

22. Vygen, J.: Near-optimum global routing with coupling, delay bounds, and power consumption. In: Nemhauser, G., Bienstock, D. (eds.) Integer Programming and Combinatorial Optimization; Proceedings of the 10th International IPCO Conference; LNCS 3064, pp. 308–324. Springer, Berlin (2004)
23. Young, N.E.: Randomized rounding without solving the linear program. In: Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms, pp. 170–178 (1995)
24. Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, pp. 538–546 (2001)