Achieving Real-Time Spectrum Sharing in 5G Underlay Coexistence with Channel Uncertainty

Shaoran Li, *Student Member, IEEE,* Yan Huang, *Member, IEEE,* Chengzhang Li, *Student Member, IEEE,* Y. Thomas Hou, *Fellow, IEEE,* Wenjing Lou, *Fellow, IEEE,* Brain A. Jalaian, *Member, IEEE,* and Stephen Russell

Abstract—Underlay coexistence is a spectrum efficient mechanism to roll out 5G picocells within a macrocell on the same spectrum. Due to a lack of cooperation between the primary users (PUs) in the macrocell and secondary users (SUs) in the picocells, it is impossible to have complete knowledge of channel conditions between them. Under such a circumstance, chance-constrained programming (CCP) has been shown to be an ideal optimization tool to address such a channel uncertainty. However, solutions to CCP are computationally intensive and cannot meet 5G's timing requirement (125 μs). To address this problem, we propose a novel scheduler called GPU-based Underlay Coexistence (GUC) with the goal of finding an approximate solution to CCP in real-time. The essence of GUC is to decompose the original optimization problem into a large number of small subproblems that are suitable for parallel computation on GPU platforms. By selecting a subset of promising subproblems and solving them in parallel with fast algorithms, we are able to leverage GPU parallel computing and develop a real-time solution. Through extensive experiments, we show that GUC meets the 125 μs requirement while achieving 90% optimality on average.

Index Terms—5G, spectrum sharing, underlay, coexistence, channel uncertainty, real-time, GPU, chance-constrained programming, scheduling, power control.

1 Introduction

Underlay coexistence is an innovative technique to roll out 5G deployment for picocells. It achieves spectrum efficiency by having secondary users (SUs) share the same spectrum with the primary users (PUs) [2]. A typical scenario is to deploy picocells within a macrocell, where the users in the macrocell and picocells are considered as PUs and SUs respectively [3]. To achieve harmonious coexistence, SUs should control their transmission powers such that the aggregated interference to each PU is below a given threshold. Under such a model, knowledge of interference channel gains from the SUs to the PUs is needed for power control. However, in the absence of feedback from the PUs, the SUs can only measure these interference channel gains based on overhearing known signals (e.g., pilot signals) from the PUs' transmissions and channel reciprocity. As a result, accurate knowledge of instantaneous interference channel gains and their distributions are hardly available. Thus, one can at best obtain limited knowledge of these interference channel gains through long-term measurements (continuous tracking) and work with their estimated means, covariances, and boundaries, etc.

Under this circumstance, there are two applicable approaches: worst-case optimization and chance-constrained programming (CCP). It is well-known that the performance of worst-case optimization is overly conservative since it only focuses on the rarely appeared worst cases (usually associated with certain boundaries) [4]. In contrast, CCP resolves this issue by allowing occasional violations of the interference threshold as long as such threshold violations happen below a small probability. CCP exploits the fact that in reality, occasional threshold violations are usually not fatal or even tolerable by the PUs for at least three reasons. First, when such violations occur, the error correction codes (e.g., Turbo code and LDPC code) at the physical layer may recover the transmitted bits that are in error (to some extent) [5]. Second, for media-rich applications (e.g., audio and video streaming), error control and concealment techniques at the application layer are widely available in existing multimedia standards [6]. Third, human perception systems (hearing and vision) are quite resilient to occasional transmission errors or packet losses [7]. By exploiting these possibilities and allowing some occasional violations, CCP promises to achieve much higher spectrum efficiency in the presence of channel uncertainty.

In this paper, we employ CCP to study underlay coexistence between 5G picocells (for SUs) and a macrocell (for PUs) on the same spectrum [8, 9] with strict timing requirement [10]. Specifically, we are interested in maximizing the spectrum efficiency of SUs in picocells while ensuring the violation probability of PUs' interference threshold stays below a given bound. This means that we need to properly allocate the sub-channels to the SUs and control their transmission powers, which is nontrivial in the presence of channel uncertainty.

Further, we must solve the optimization problem within 125 μs, the smallest scheduling time interval in 5G physical layer transmission [11, 12]. Though there are several prior efforts employing CCP to study underlay problems (see,

Manuscript received March 4, 2021; revised August 29, 2021; accepted October 7, 2021. Date of publication XXX, 2021. Some early results of this research were supported in part by U.S. Army Research Laboratory under Cooperative Agreement No. W911NF1820293. The work of Y.T. Hou was also supported in part by NSF under grant 1642873, Virginia Commonwealth Cyber Initiative (CCI), and Virginia Tech Institute for Critical Technology and Applied Science (ICTAS). An abridged version of this paper appeared in the Proc. IEEE GLOBECOM, Waikoloa, HI, Dec. 2019 [1].

S. Li, C. Li, Y.T. Hou, and W. Lou are with Virginia Tech, Blacksburg, VA 24061.
Y. Huang is with NVIDIA Corp., Santa Clara, CA 95051.
B.A. Jalaian and S. Russell are with CCDC Army Research Laboratory, Adelphi, MD 20783.

e.g., [13–16]), their typical running times are from hundreds of milliseconds to tens of seconds, which are orders of magnitude larger than the 125 μs real-time requirement. Note that the term "real-time" in this paper refers to the actual running time (as measured by a wall clock) of an algorithm on a hardware platform. Under such a quantitative objective measure, traditional complexity analysis (in the form of big $O(\cdot)$) will not be much useful as it is not directly tied to wall-clock time. Even an algorithm with $O(1)$ complexity may still have a computation time larger than our timing requirement.

To address this real-time challenge, we propose a novel parallel algorithm and implement it on GPU platforms. Though GPU has been used in other problems to provide strict timing guarantee such as macrocell Proportional Fair (PF) scheduler [17] and MIMO beamforming [18], these GPU solutions cannot be directly applied to our problem due to some fundamental differences in problem settings and mathematical structures. To the best of our knowledge, there is no work to date that uses GPU to address 5G underlay coexistence problem via CCP in order to meet the 125 μs timing requirement.

The main contributions of this work are summarized as follows:

- We propose and implement a real-time solution for 5G underlay coexistence that meets the 125 μs requirement in 5G standards. Our proposed solution maximizes the spectrum efficiency through scheduling and power control of SUs while keeping the probability of violating PUs' interference threshold under a given target.
- Our proposed solution – GPU-based Underlay Coexistence (GUC) first decomposes the original problem into a large number of subproblems. Instead of solving all the subproblems, we only select a subset of promising subproblems where each subproblem corresponds to a fixed sub-channel allocation among SUs. Then these selected subproblems are solved in parallel based on closed-form starting points and scaling-based local search. Finally, the feasible solution (sub-channel allocations and transmission powers of SUs) with the highest objective value will be chosen as the final solution.
- We implement our GUC algorithm on an off-the-shelf Nvidia Telsa V100 GPU using CUDA programming tool. To conserve computation time, we optimize each step in our implementation, including proper thread allocations, minimizing memory access time, and employing parallel reduction, etc.
- Through comprehensive experiments, we demonstrate that GUC maximizes the spectrum efficiency of SUs in real-time while guaranteeing probabilistic violation of PUs' interference threshold. Specifically, our measured GUC computation time meets the 125 μs timing requirements, which is at least 10^4 times faster than commercial solvers (e.g., Gurobi on CPU). GUC is also able to achieve 90% optimality on average.

We organize the remainder of this paper as follows. In Section 2, we review related work. In Section 3, we introduce the 5G underlay coexistence model and formulate the optimization problem. In Section 4, we present our algorithm design. In Section 5, we present how we implement our algorithm on GPU platforms. In Section 6, we present results from our experiments. Section 7 concludes this paper.

2 Related Work

We review related work on CCP and GPU, with a focus on their applications in wireless communications.

CCP CCP is typically used to address uncertainties and provide probabilistic guarantees. Existing CCP works in wireless communications have studied problems in underlay coexistence [13–16], OFDM scheduling [19], D2D communication [20, 21], and QoS guarantee in wireless video streaming [22]. However, a common issue with these works is that the computation time of the proposed solutions is orders of magnitude larger than the "real-time" requirement in our problem, i.e., 125 μs. So none of the proposed solutions are useful to address our problem.

GPU GPU has been used to accelerate computation time due to its massive parallel processing capability. In wireless communications, GPU has been applied to PF scheduling [17] at a base station (BS), MIMO beamforming [18], Age of Information (AoI) scheduling [23], channel coding [24], and signal processing [25, 26]. Most notably, in [17], we developed a GPU-based solution for 5G PF scheduler that consists of problem decomposition, selection of a subset of subproblems, and solving these subproblems in parallel. These ideas laid the groundwork on how to apply GPU to address complex optimization problems in real-time. However, each problem is unique and the detailed design in each of these steps is not trivial and must be carefully crafted for the underlying problem based on its unique mathematical structures and physical meanings. Likewise, the implementation efforts of a proposed algorithm must be custom designed otherwise it will unlikely be able to meet the desired timing requirement.

3 System Model and Problem Formulation

Consider a two-tier setting where several 5G picocells are deployed inside a macrocell following an underlay coexistence (see Fig. 1). Users in the macrocell and picocells are considered as PUs and SUs respectively. Both the macrocell and the picocells share the same spectrum. The BS in a picocell is likely to be deployed as part of a set-up box inside a residential unit [27, 28]. Since multiple picocells are sharing the same spectrum with the macrocell and the number of users in a picocell is much fewer that that in a macrocell, it is therefore reasonable to have a picocell to occupy only a fraction of the macrocell's spectrum in a non-overlapping fashion from neighboring picocells (to avoid inter-picocell interference). This scheme is known as *fractional frequency reuse* in the literature (see, e.g., [29, 30]).

For a specific picocell, due to its small footprint, there can only be a few PUs nearby. In underlay coexistence, the burden of interference control solely rests upon the secondary network, i.e., picocells in our setting. To differentiate multiple PUs in the absence of any explicit cooperation, the SUs can exploit the orthogonal pilots that they hear

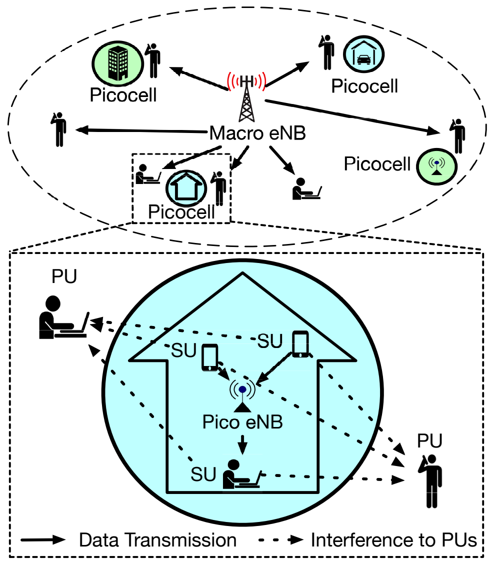


Fig. 1. Network topology of picocells within one macrocell

TABLE 1
Notations

Symbol	Definition
g_{ij}^m	Interference channel gain from SU i to PU j on RB m
$\mathbf{g}_j$	A column vector: $\left[g_{1j}^1, g_{1j}^2, \cdots, g_{1j}^M, g_{2j}^1, \cdots, g_{Nj}^M\right]^T$
$\overline{\mathbf{g}}_j$	Mean of channel gain vector $\mathbf{g}_j$
$h_{i\text{B}}^m$	Transmission channel gain from SU i to its pico BS on RB m
I_j	Interference threshold for PU j
J	Number of nearby PUs
$\mathcal{J}$	The set of integers from 1 to J: $\{1, 2, 3, \cdots, J\}$
M	Number of RBs for transmission in the picocell
$\mathcal{M}$	The set of integers from 1 to M: $\{1, 2, 3, \cdots, M\}$
N	The number of SUs in the picocell
$\mathcal{N}$	The set of integers from 1 to N: $\{1, 2, 3, \cdots, N\}$
$p_{i\text{B}}^m$	Transmission power from SU i to pico BS on RB m
$\mathbf{p}$	A column vector: $\left[p_{1\text{B}}^1, p_{1\text{B}}^2, \cdots, p_{1\text{B}}^M, p_{2\text{B}}^1, \cdots, p_{N\text{B}}^M\right]^T$
$P_i^{\max}$	Maximum transmission power of SU i over all RBs
$\mathbf{R}_j$	Covariance matrix of channel gain vector $\mathbf{g}_j$
w_i	Weight of SU i
$x_{i\text{B}}^m$	A binary variable indicating whether or not SU i transmits on RB m
ϵ_j	Risk level (probability upper bound) of violating PU j's interference threshold I_j

from the PUs, or some location techniques based on existing spectrum sensing algorithms [31, 32].

It is easy to see that the most challenging interference control scenario occurs when the SUs are transmitting to the pico BS (uplink) and the PUs are receiving from the macro BS (downlink). We will focus on this scenario in this paper and our proposed solution can be easily extended to other transmission scenarios. Each picocell operates in a time-slotted system and the time domain is divided into small Transmission Time Intervals (TTIs). In each TTI, a pico BS gathers the channel conditions from the SUs and then runs a scheduling algorithm for the SUs' resource allocation and power control. These decisions will be sent to the SUs through the control channel immediately and will be applied for SUs' uplink transmissions in the next TTI. Thus, the time for running the scheduling algorithm at the pico BS is at most one TTI. Since the neighboring picocells are operating independently on non-overlapping frequencies, we will focus on one picocell and maximize the spectrum efficiency for the SUs inside the picocell while carefully control their interference to each nearby PU. Under CCP, this means that the violation probability of PUs' interference thresholds will not exceed a given bound.

The notations used in this paper are summarized in Table 1. Let N and J denote the number of SUs in the picocell and its nearby PUs, respectively. Suppose the transmission bandwidth allocated to this picocell is divided into M sub-channels. Based on cellular terminology, we call each sub-channel in one TTI as a resource blocks (RB). Clearly, there are M RBs per TTI. We use the weighted sum of channel capacity as our objective for each TTI. This gives much flexibility to define fairness by the operator. For example, if the weights are static and assigned to the SUs once and for all (no change from TTI to TTI), then this is equivalent to assigning priority to each user. As another (more sophisticated) example, if the weights are dynamically changed per TTI, such as based on a SU's long-term data rate, then this is equivalent to the well-known Proportional Fair (PF) scheduler [17]. Recall that our problem is to design a scheduling algorithm that runs within each TTI to determine how RBs are to be allocated and the corresponding transmission powers for the SUs in the next TTI. In other words, the available running time for the proposed scheduling algorithm must be no greater than one TTI (as low as 125 μs in 5G).

For the considered TTI,[1] denote a binary variable $x_{i\text{B}}^m$ as whether or not SU i will transmit on RB m after scheduling, i.e.,

$$x_{i\text{B}}^m = \begin{cases} 1 & \text{if SU } i \text{ will transmit on RB } m\ , \\ 0 & \text{otherwise}\ . \end{cases} \tag{1}$$

Here "B" stands for the pico BS.

In single user orthogonal frequency-division multiple access (OFDMA), each RB can be assigned to at most one SU. Thus, we have constraints for RB allocations given as

$$\sum_{i \in \mathcal{N}} x_{i\text{B}}^m \le 1 \qquad (m \in \mathcal{M})\ , \tag{2}$$

where $\mathcal{M}$ denotes the set of RBs $\{1, 2, \cdots, M\}$.

Denote $p_{i\text{B}}^m$ as the transmission power of SU i on RB m and let $P_{i\text{B}}^{\max}$ represents the total transmission power limit of SU i across all RBs. The constraints for scheduling transmission powers and device power limits are given by

$$0 \le p_{i\text{B}}^m \le x_{i\text{B}}^m P_{i\text{B}}^{\max} \qquad (i \in \mathcal{N},\ m \in \mathcal{M})\ , \tag{3}$$

$$\sum_{m \in \mathcal{M}} p_{i\text{B}}^m \le P_{i\text{B}}^{\max} \qquad (i \in \mathcal{N})\ . \tag{4}$$

where $\mathcal{N}$ is the set $\{1, 2, \cdots, N\}$.

Denote g_{ij}^m as the interference channel gain from SU i to PU j on RB m and let I_j be the interference threshold for PU j. Under CCP, the interference control from the SUs to the PUs can be formulated as chance constraints, given by

$$\mathbb{P}\left\{\sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}} g_{ij}^m p_{i\text{B}}^m \le I_j\right\} \ge 1 - \epsilon_j \qquad (j \in \mathcal{J})\ , \tag{5}$$

1. For ease of exposition, we drop the notation of time (for each TTI) when there is no confusion.

where $\mathcal{J} = \{1, 2, \cdots, J\}$, $\mathbb{P}\{\cdot\}$ denotes probability, and ϵ_j is the *risk level* (probability upper bound) for violation of interference threshold I_j. Clearly, a higher ϵ_j leads to a larger tolerance to violation of interference threshold I_j and hence higher spectrum efficiency. A typical value of ϵ_j ranges from 0.01 to 0.5 depending on the requirement of PU j. In (5), $p_{i\mathrm{B}}^m$ is an optimization variable while interference channel gain g_{ij}^m is modeled as a random variable. Constraints (5) state that the aggregate interference from the SUs to PU j over all RBs should stay below threshold I_j with a probability no smaller than $1-\epsilon_j$. For generality, we assume only the mean and covariance of g_{ij}^m's are known since these statistics are rather time-invariant compared to the instantaneous values of g_{ij}^m's.

For constraints (5), we can rewrite it in the matrix form

$$\mathbb{P}_{\mathbf{g}_j \sim (\overline{\mathbf{g}}_j, \mathbf{R}_j)} \left\{ \mathbf{g}_j^T \mathbf{p} \le I_j \right\} \ge 1 - \epsilon_j \qquad (j \in \mathcal{J}) , \qquad (6)$$

where superscript "T" denotes transposition. $\mathbf{p}$ is an $MN \times 1$ column vector consists of MN transmission powers from the SUs (over all RBs) to the pico BS, and is given as

$$\mathbf{p} = \left[p_{1\mathrm{B}}^1, \cdots, p_{1\mathrm{B}}^M, p_{2\mathrm{B}}^1, \cdots, p_{2\mathrm{B}}^M, \cdots, p_{N\mathrm{B}}^1, \cdots, p_{N\mathrm{B}}^M\right]^T . \quad (7)$$

$\mathbf{g}_j$ is an $MN \times 1$ random column vector, and is defined as

$$\mathbf{g}_j = \left[g_{1j}^1, \cdots, g_{1j}^M, g_{2j}^1, \cdots, g_{2j}^M, \cdots, g_{Nj}^1, \cdots, g_{Nj}^M\right]^T , \quad (8)$$

which represents MN random interference channel gains from the SUs (over all RBs) to PU j. $\overline{\mathbf{g}}_j$ (a $MN \times 1$ column vector) and $\mathbf{R}_j$ (a $MN \times MN$ matrix) are the known mean and covariance of $\mathbf{g}_j$, respectively.

By normalizing the bandwidth of each RB to 1 unit, we have the following problem formulation

$$\begin{aligned} \text{(P1)} \quad \max_{x_{i\mathrm{B}}^m, p_{i\mathrm{B}}^m} \quad & \sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}} w_i \log_2(1 + h_{i\mathrm{B}}^m p_{i\mathrm{B}}^m) \\ \text{s.t.} \quad & \text{RB allocations } (2) , \\ & \text{Uplink transmission powers } (3) , \\ & \text{Device power limits } (4) , \\ & \text{Violation probability guarantees } (6) , \\ & x_{i\mathrm{B}}^m \in \{0,1\}, \;\; p_{i\mathrm{B}}^m \ge 0 , \end{aligned}$$

where w_i is the given weight of SU i in current TTI. $h_{i\mathrm{B}}^m$ is the transmission channel gain of SU i toward the pico BS on RB m, which includes the effects of interference from the macro BS to the pico BS and thermal noise at the pico BS.

Though we know the mean and covariance of $\mathbf{g}_j$, its distribution has an infinite number of possibilities. Therefore, P1 is intractable due to chance constraints (6).

4 GUC: A Novel Parallel Algorithm

In this section, we present a novel scheduling and power control algorithm to solve P1. Our whole design for P1 is called GUC (short for GPU-based Underlay Coexistence), which consists of two parts: GUC algorithm design in this section and the corresponding implementation efforts in Section 5.

4.1 Reformulation of chance constraints

The main difficulty of P1 lies in chance constraints (6). Thus, a necessary step to solve P1 is to substitute (6) with deterministic constraints and obtain a deterministic optimization problem. The state-of-the-art technique to perform such a substitution is called *Exact Conic Reformulation* (ECR) [16], which is able to replace the intractable chance constraints (6) with convex deterministic constraints without any relaxations. Comparing with other approaches such as Chebyshev inequality and Bernstein Approximation, ECR requires fewer assumptions and offers better performance. Using ECR, it can be shown that constraints (6) are mathematically equivalent (w.r.t. $\mathbf{p}$) to the following deterministic constrains [16]

$$\sqrt{\frac{1-\epsilon_j}{\epsilon_j}} \sqrt{\mathbf{p}^T \mathbf{R}_j \mathbf{p}} + \overline{\mathbf{g}}_j^T \mathbf{p} \le I_j \qquad (j \in \mathcal{J}) . \qquad (9)$$

ECR guarantees that the optimization space of $\mathbf{p}$ in (9) is the same as that of (6), which means this reformulation is exact. It also means each equality in (9) is achievable for some distributions of $\mathbf{g}_j$ where the threshold violation probability is exactly ϵ_j. For all other distributions of $\mathbf{g}_j$, the threshold violation probability will be smaller than ϵ_j. We refer interested readers to [16] for more details of ECR.

Replacing (6) with (9) in P1, we have a deterministic maximization problem as the following

$$\begin{aligned} \text{(P2)} \quad \max_{x_{i\mathrm{B}}^m, p_{i\mathrm{B}}^m} \quad & \sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}} w_i \log_2(1 + h_{i\mathrm{B}}^m p_{i\mathrm{B}}^m) \\ \text{s.t.} \quad & \text{RB allocations } (2) , \\ & \text{Scheduling transmission powers } (3) , \\ & \text{Device power limits } (4) , \\ & \text{Interference power control from ECR } (9) , \\ & x_{i\mathrm{B}}^m \in \{0,1\}, p_{i\mathrm{B}}^m \ge 0 . \end{aligned}$$

P2 belongs to MINLP. One could try to use a commercial solver such as Gurobi to solve P2. But we find that its computational time is on the order of seconds (see Section 6) and thus it cannot meet the 125 μs timing requirement. The fundamental issue is that they require sequential iterations, which is time-consuming.

4.2 GUC Algorithm Design

Our algorithm design is to exploit parallelism to reduce computation time. The general idea of solving a complex optimization problem is to decompose the original problem into a large number of small subproblems that can be solved in parallel. If the number of subproblems is too large to be solved in parallel, we will need to select a subset of these subproblems to solve (with perhaps some loss in performance). After the subset of subproblems is solved, we can pick the best feasible solution in the hope that its performance is close to the optimal solution.

Although the main idea of GUC algorithm design is not hard to understand, how to accomplish each step is far from trivial and constitutes the main efforts in our algorithm design.

4.2.1 *Decomposition of P2*

In this section, we show how to decompose P2 into parallel subproblems. In P2, there are two sets of decision variables: the binary RB allocations $x_{i\mathrm{B}}^m$'s and the continuous transmission powers $p_{i\mathrm{B}}^m$'s. Clearly, $x_{i\mathrm{B}}^m$'s are dominant variables and should be considered first. It is also easier to perform decomposition since $x_{i\mathrm{B}}^m$'s are binary variables.

Our decomposition of P2 is based on enumerating all feasible RB allocations and correspondingly setting $x_{i\mathrm{B}}^m$'s values to 0 or 1. Once $x_{i\mathrm{B}}^m$'s are fixed under a feasible RB allocation, we have a subproblem instance from P2 that involves only continuous variables $p_{i\mathrm{B}}^m$'s. By (2) (i.e., a RB can only be allocated to at most one SU), a feasible solution should have no more than one non-zero element in set $\{x_{1\mathrm{B}}^m, x_{2\mathrm{B}}^m, \cdots, x_{N\mathrm{B}}^m\}$. Denote π_m as the SU that is allocated with RB m. Then we have $\pi_m \in \mathcal{N}$ for $m \in \mathcal{M}$; $x_{i\mathrm{B}}^m = 1$ if $i = \pi_m$ and $x_{i\mathrm{B}}^m = 0$ otherwise. Denote a $1 \times M$ row vector $\boldsymbol{\pi} = \{\pi_1, \pi_2, \cdots, \pi_M\}$ as the set of SUs to which the M RBs are assigned. Clearly, for $m \in \mathcal{M}$, $p_{i\mathrm{B}}^m > 0$ if $i = \pi_m$ and $p_{i\mathrm{B}}^m = 0$ otherwise. Note that some of the elements in $\boldsymbol{\pi}$ can be the same, meaning that multiple RBs are allocated to a SU. Thus, an instance of subproblem (for a given $\boldsymbol{\pi}$) is:

$$(\text{P3}) \ \max_{p_{\pi_m\mathrm{B}}^m} \sum_{m \in \mathcal{M}} w_{\pi_m} \log_2(1 + h_{\pi_m\mathrm{B}}^m p_{\pi_m\mathrm{B}}^m)$$

$$\text{s.t.} \sum_{m \in \mathcal{M}, \pi_m = i} p_{\pi_m\mathrm{B}}^m \le P_{i\mathrm{B}}^{\max} \quad (i \in \pi) \ , \tag{10}$$

$$\sqrt{\frac{1-\epsilon_j}{\epsilon_j}} \sqrt{\mathbf{p}_\pi^T \mathbf{R}_{\pi j} \mathbf{p}_\pi} + \overline{\mathbf{g}}_{\pi j}^T \mathbf{p}_\pi \le I_j \ (j \in \mathcal{J}) \ , \tag{11}$$

$$p_{\pi_m\mathrm{B}}^m \ge 0 \qquad (m \in \mathcal{M}) \ . \tag{12}$$

The $M \times 1$ column vector $\mathbf{p}_\pi = [p_{\pi_1 B}^1, p_{\pi_2 B}^2, \cdots, p_{\pi_m\mathrm{B}}^M]^T$ represents the transmission powers corresponding to a given RB allocation $\boldsymbol{\pi}$. Denote a $M \times 1$ random column vector $\mathbf{g}_{\pi j} = [g_{\pi_1 j}^1, g_{\pi_2 j}^2, \cdots, g_{\pi_m j}^M]^T$, then let $\overline{\mathbf{g}}_{\pi j}$ (a $M \times 1$ column vector) and $\mathbf{R}_{\pi j}$ (a $M \times M$ matrix) be the mean and covariance of $\mathbf{g}_{\pi j}$ respectively. Clearly, $\overline{\mathbf{g}}_{\pi j}$ and $\mathbf{R}_{\pi j}$ are truncated from $\overline{\mathbf{g}}_j$ and $\mathbf{R}_j$. Such truncation is performed by removing those elements in $\overline{\mathbf{g}}_j$ and $\mathbf{R}_j$ corresponding to the positions of zero values in $\mathbf{p}$. After truncation, it is easy to show the following result hold:

$$\mathbf{p}_\pi^T \mathbf{R}_{\pi j} \mathbf{p}_\pi = \mathbf{p}^T \mathbf{R}_j \mathbf{p} \ , \ \overline{\mathbf{g}}_{\pi j}^T \mathbf{p}_\pi = \overline{\mathbf{g}}_j^T \mathbf{p} \ . \tag{13}$$

Given (13), constraints (10) and (11) are equivalent to constraints (4) and (9) respectively as we only deal with those non-zero transmission powers $p_{\pi_m\mathrm{B}}^m$'s. Thus, a feasible solution to P3 is also a feasible solution to P2 and P1.

4.2.2 *Select Subproblems*

In our original problem, there are M RBs and each RB has N possible RB allocations, shown as a column of N green squares in the left part of Fig. 2. Thus, the total number of possible $\boldsymbol{\pi}$ is N^M (enumerating all feasible assignments of $x_{i\mathrm{B}}^m$'s). Due to our strict timing requirement, we can only compute a subset of subproblems in parallel. Denote K_x as the number of selected subproblems, where $K_x \ll N^M$. In the rest of this section, we show how to select these K_x subproblems by identifying a promising subproblem space and random sampling inside this space.

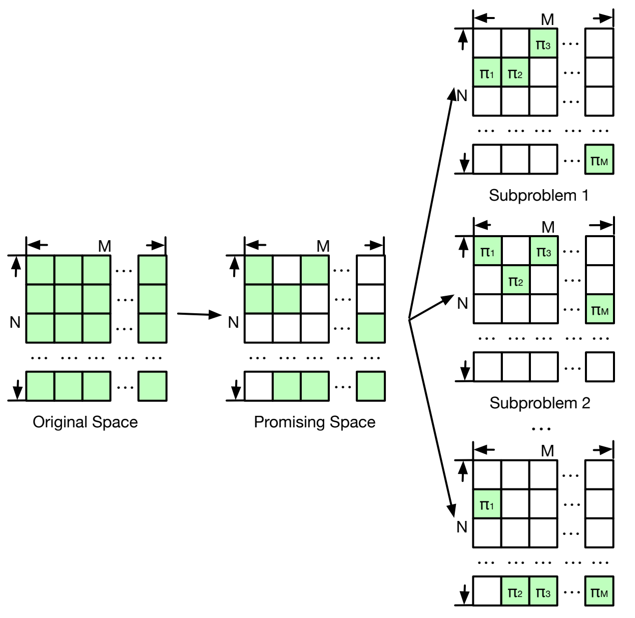


Fig. 2. Select K_x subproblems by random sampling in a promising space

Identifying a Promising Space A promising space can be defined as a subspace (within the original problem space) that contains a set of subproblems with reasonably good (acceptable) solutions. It is possible that the optimal solution may fall outside this promising space. But as long as the best feasible solution inside this promising space is reasonably good (acceptable), it will serve our purpose.

To identify a promising space, we propose to limit the number of possible RB allocations for each RB from N to a smaller number, denoted as D, i.e., $D < N$. Thus, we need to design a metric to "measure" how good an RB allocation is before solving the problem. Clearly, this metric should be based on the given parameters such as w_i's, $h_{i\mathrm{B}}^m$'s and g_{ij}^m's. Intuitively, we can consider allocating RB m to SU i a good allocation if w_i and $h_{i\mathrm{B}}^m$ are relatively high, and g_{ij}^m is relatively small. To exploit this idea, we define a metric $d_{i\mathrm{B}}^m$ to evaluate each scheduling decision $x_{i\mathrm{B}}^m$ as following

$$d_{i\mathrm{B}}^m = \frac{w_i h_{i\mathrm{B}}^m}{u_{ij}^m} \quad (i \in \mathcal{N}, \ m \in \mathcal{M}) \ , \tag{14a}$$

$$u_{ij}^m = \sum_{j \in \mathcal{J}} \left\{ \sqrt{\frac{1-\epsilon_j}{\epsilon_j}} \sqrt{||(\mathbf{R}_j)_{((i-1)M+m)*}||_1} + \overline{g}_{ij}^m \right\} \ . \tag{14b}$$

where $(\mathbf{R}_j)_{((i-1)M+m)*}$ is the $((i-1)M+m)$-th column of $\mathbf{R}_j$, $||\cdot||_1$ is the L_1-norm, and $\overline{g}_{ij}^m$ is the mean of g_{ij}^m. In (14a), $w_i h_{i\mathrm{B}}^m$ is the gradient of the objective function w.r.t. $p_{i\mathrm{B}}^m$ at $p_{i\mathrm{B}}^m = 0$; u_{ij}^m is related to the mean of interference channel gain $\overline{g}_{ij}^m$, which has a similar form with (9). Taking both into considerations, we use a simple division to define $d_{i\mathrm{B}}^m$ and obtain the metric in (14a).

Then the SUs $s_1^m, s_2^m, \cdots, s_D^m$ are considered promising for RB m's allocation. Denote a set $\mathcal{S}_D^m = \{s_1^m, s_2^m, \cdots, s_D^m\}$ and RB m can only be allocated to the SUs in set $\mathcal{S}_D^m$, i.e., $\pi_m \in \mathcal{S}_D^m$ (instead of $\mathcal{N}$ previously). This process of identifying a promising set is shown in the middle part of Fig. 2 where each column (corresponding to one RB) only has D possible allocated SUs (instead of N). We perform the above procedure for each RB in parallel and obtain its corresponding promising set $\mathcal{S}_D^m$. As each RB only has D possible allocations, we have a promising space $\mathcal{S}$ consisting of D^M subproblems, i.e.,

$$\boldsymbol{\pi} \in \mathcal{S} = \mathcal{S}_D^1 \times \mathcal{S}_D^2 \times \cdots \times \mathcal{S}_D^M \ , \tag{15}$$

where "$\times$" denotes Cartesian product.

Clearly, a larger D expands the promising space while a smaller D shrinks the promising space. We propose to determine a suitable value of D based on historical results that can be easily applied in practice. To determine a suitable D, we first need to solve P2 in the original space for the optimal solution. Then we solve P2 optimally in the promising space under different D's and compare the objective ratios with the optimal solution. The objective ratio is a non-decreasing function of D. By assessing the objective ratios in the promising space, we can pick the smallest D that offers acceptable (or near-optimal) performance.

Random Sampling If $K_x < D^M$, we need to perform sampling to obtain K_x subproblems. Otherwise, we can skip this step. At this point, there are D^M events in the promising space $\mathcal{S}$ and each event corresponds to one subproblem instance. For an event $\varphi = \{\varphi_1, \varphi_2, \cdots, \varphi_M\} \in \mathcal{S}$, we define $d_\varphi = d^1_{\varphi_1 B} \cdot d^2_{\varphi_2 B} \cdots d^M_{\varphi_M B}$. Then we assign event φ with the following probability

$$\mathbb{P}\{\varphi\} = \frac{d_\varphi}{\sum\limits_{\phi \in S} d_\phi} \quad \text{for } \varphi \in \mathcal{S} \ . \tag{16}$$

The denominator in (16) is to normalize d_φ so that it is a valid probability assigned to event φ. We then perform K_x samples following the probabilities given in (16) and obtain K_x events (subproblems). As shown in the right part of Fig. 2, we obtain K_x subproblems and each subproblem has fixed RB allocations to the SUs.

4.2.3 Solve the Subproblems

In this subsection, we will show how to solve each subproblem P3 within a limited number of operations to reduce computation time. The subproblems are independent among each other and hence we can solve the subproblems in parallel without any information exchange. Due to the same mathematical structure of the subproblems, we only need to design one algorithm for all sub-problems, i.e., single-instruction multiple data (SIMD). In a subproblem P3, there are M continuous decision variables $p^m_{\pi_m \mathrm{B}}$'s and P3 is a convex optimization problem. A conventional approach to solving such a convex problem is by gradient-based iterations. However, such an approach usually involves tens of iterations to converge and cannot meet the 125 μs timing requirement. To overcome this challenge, we have to sacrifice optimality in favor of approximate solutions that are fast and have performance close to that of the optimal solution.

We notice that for the three set of constraints (10)–(12) in P3, constraints (12), the nonnegativity of decision variables $p^m_{\pi_m \mathrm{B}}$'s, are easy to meet and thus we should mainly focus on constraints (10) and (11). Since these two constraints only have linear and quadratic terms regarding $p^m_{\pi_m \mathrm{B}}$'s, we propose to solve P3 based on decent starting points and a simple scaling-based local search, given as below.

Starting Points A starting point of P3 consists M transmission powers for the scheduled SUs, which is in the form of $\{p^1_{\pi_1 \mathrm{B}}, p^2_{\pi_2 \mathrm{B}}, \cdots, p^M_{\pi_M \mathrm{B}}\}$. For good performance, we use K_p starting points (each with M transmission powers) for each subproblem. Note that this means we have a total of $K_x K_p$ starting points since we have chosen K_x subproblems in the previous step. Since we can handle those starting points in parallel, this will only slightly increase the computation time.

For a specific subproblem (with fixed $\boldsymbol{\pi}$), the k-th ($k = 1, 2, \cdots, K_p$) starting point contains M transmission powers, in the form of $\mathbf{p}^k_\pi = \left[p^{1k}_{\pi_1 \mathrm{B}}, p^{2k}_{\pi_2 \mathrm{B}}, \cdots, p^{mk}_{\pi_m \mathrm{B}}, \cdots, p^{Mk}_{\pi_M \mathrm{B}}\right]^T$. We set $p^{mk}_{\pi_m \mathrm{B}}$ as

$$p^{mk}_{\pi_m \mathrm{B}} = \frac{w_{\pi_m} h^m_{\pi_m \mathrm{B}}}{\sum\limits_{m \in \mathcal{M}} w_{\pi_m} h^m_{\pi_m \mathrm{B}}} \cdot \frac{\sum\limits_{j \in \mathcal{J}} I_j}{\sum\limits_{j \in \mathcal{J}} u^m_{\pi_m j}} + \delta v^m_k \tag{17a}$$

$$= \frac{d^m_{\pi_m \mathrm{B}} \sum\limits_{j \in \mathcal{J}} I_j}{\sum\limits_{m \in \mathcal{M}} w_{\pi_m} h^m_{\pi_m \mathrm{B}}} + \delta v^m_k \qquad (m \in \mathcal{M}) \ , \tag{17b}$$

where v^m_k is a uniformly distributed random variable and δ is a small positive number that ensures all the K_p starting points are within a sphere. The first fraction in (17a) is proportional to the weight w_{π_m} and transmission channel gain $h^m_{\pi_m \mathrm{B}}$ while the second fraction in (17a) is the average transmission power for all SUs regarding interference control. Note that the intermediate results of $d^m_{\pi_m \mathrm{B}}$ and $w_{\pi_m} h^m_{\pi_m \mathrm{B}}$ are already calculated in (14) during the process of finding a promising space. Therefore, (17b) is easy to implement since I_j and δ are given constants.

However, the transmission powers given in the starting points (17) have no guarantee of feasibility nor performance. Thus, GUC uses scaling-based local search to refine each starting point from (17) to guarantee feasibility while improving performance if possible, as detailed below.

Scaling-based Local Search For simplicity, we drop the notation of k in the rest of this section. This local search is done by three scaling procedures regarding (10), (11) and (10) again, described as the following.

First scaling based on (10). The i-th constraint in (10) states that the total transmission powers from SU i cannot exceed its device limit. Thus, the idea of the first scaling is to scale down the transmission powers if the corresponding SU exceeds its device power limit.

Given the transmission power obtained from the starting points (17), we can calculate the current total transmission power from SU i as

$$P^{\text{total}}_{i\mathrm{B}} = \sum_{m \in \mathcal{M},\ \pi_m = i} p^m_{\pi_m \mathrm{B}} \quad (i \in \pi) \ . \tag{18}$$

For SU i, if $P^{\text{total}}_{i\mathrm{B}} > P^{\max}_{i\mathrm{B}}$, we scale down the transmission powers from SU i such that the total transmission power from SU i is equal to its device limit. Otherwise, we don't do anything. Thus, the first scaling for $p^m_{\pi_m \mathrm{B}}$ is summarized as

$$p^m_{\pi_m \mathrm{B}} = p^m_{\pi_m \mathrm{B}} \cdot \min\left\{1, \frac{P^{\max}_{\pi_m \mathrm{B}}}{P^{\text{total}}_{\pi_m \mathrm{B}}}\right\} \quad (m \in \mathcal{M}) \ . \tag{19}$$

After the first scaling, (10) is guaranteed to be feasible since no SU will exceed its device limit. Let $\boldsymbol{\pi}' = \{i : i \in \mathcal{N}, P_{i\text{B}}^{\text{total}} = P_{i\text{B}}^{\text{max}}\}$ be the set of SUs who has already reached their device limit, which will be used in the second scaling.

Second scaling based on (11). The j-th constraint in (11) represents the transmission opportunities of SUs allowed by PU j. Notice that the objective function monotonically increases regarding the transmission powers, so we would like to increase these transmission powers while ensuring constraints (11) hold. Thus, in the second scaling, we will not only focus on the feasibility of constraints (11) but also try to improve the objective value by increasing the transmission powers, if possible.

To derive the second scaling based on constraints (11), we calculate the current interference level from the SUs in $\boldsymbol{\pi}$ (denoted as $\hat{I}_j$) and the average interference from the SUs in $\boldsymbol{\pi}'$ (denoted as $\hat{I}'_j$), given as

$$\hat{I}_j = \sqrt{\frac{1-\epsilon_j}{\epsilon_j}}\sqrt{\mathbf{p}_\pi^T \mathbf{R}_{\pi j}\mathbf{p}_\pi} + \overline{\mathbf{g}}_{\pi j}^T \mathbf{p}_\pi \ , \tag{20a}$$

$$\hat{I}'_j = \sum_{\pi_m \in \pi'} \overline{g}_{\pi_m j}^m p_{\pi_m \text{B}}^m \ , \tag{20b}$$

where $\overline{g}_{\pi_m j}^m$ is the mean of $g_{\pi_m j}^m$ (the m-th element of $\overline{\mathbf{g}}_{\pi_m j}^m$). Note that $\hat{I}'_j < \hat{I}_j$ holds.

Based on $\hat{I}_j$ from (20a) and $\hat{I}'_j$ from (20b), we obtain the scaling factor λ_j based on the j-th constraint of (11) as following

$$\lambda_j = \begin{cases} \dfrac{I_j}{\hat{I}_j} & \text{if } \hat{I}_j \geq I_j \ , \\ \dfrac{I_j - \hat{I}'_j}{\hat{I}_j - \hat{I}'_j} & \text{if } \hat{I}_j < I_j \ . \end{cases} \tag{21}$$

We perform the second scaling by using the minimum of λ_j for all J scaling factors as

$$p_{\pi_m \text{B}}^m = \begin{cases} p_{\pi_m \text{B}}^m \cdot \min\{1, \lambda_1, \cdots, \lambda_J\} & \text{if } \pi_m \in \boldsymbol{\pi}' \\ p_{\pi_m \text{B}}^m \cdot \min\{\lambda_1, \cdots, \lambda_J\} & \text{otherwise} \end{cases} \tag{22}$$
$$(m \in \mathcal{M}) \ .$$

Note that there are two possible cases regarding the second scaling: the transmission powers are scaled up if $\hat{I}_j < I_j$ holds for every $j \in \mathcal{J}$ and are scaled down otherwise. If we are able to scale up the transmission powers, the objective value will increase. In other words, we are making use of the transmission opportunities allowed by the PUs. Moreover, we will only scale up the $p_{\pi_m \text{B}}^m$'s from the SUs who have not reached their device limit (from $\boldsymbol{\pi} \backslash \boldsymbol{\pi}'$) since the SUs in $\boldsymbol{\pi}'$ have already reached their transmission power limits.

In both cases, the outcome of the second scaling has the following property.

Property 1. After the second scaling based on (22), the transmission powers satisfy constraints (11).

Proof Let $\mathbf{p}_\pi$ be the transmission powers before the second scaling and define function $f_j(\mathbf{p}_\pi)$ as

$$f_j(\mathbf{p}_\pi) = \sqrt{\frac{1-\epsilon_j}{\epsilon_j}}\sqrt{\mathbf{p}_\pi^T \mathbf{R}_{\pi j}\mathbf{p}_\pi} + \overline{\mathbf{g}}_{\pi j}^T \mathbf{p}_\pi \ . \tag{23}$$

Clearly, we have $f_j(\mathbf{p}_\pi) = \hat{I}_j$ based on the definition of $\hat{I}_j$ in (20a). Since the scaling factor in the second scaling (22) is the minimum of all λ_j's, to prove Property 1, we only need to show constraints (11) are satisfied for each λ_j defined in (21). In other words, we need to show $f_j(\mathbf{q}_\pi) \leq I_j$ where $\mathbf{q}_\pi = [q_{\pi_1 \text{B}}^1, q_{\pi_2 \text{B}}^2, \cdots, q_{\pi_m \text{B}}^M]^T$ and

$$q_{\pi_m \text{B}}^m = \begin{cases} p_{\pi_m \text{B}}^m \cdot \min\{1, \lambda_j\} & \text{if } \pi_m \in \boldsymbol{\pi}' \\ p_{\pi_m \text{B}}^m \cdot \lambda_j & \text{otherwise} \end{cases} \quad (m \in \mathcal{M}) \ . \tag{24}$$

Consider the following two cases regarding λ_j.

Case 1. $0 < \lambda_j \leq 1$. This happens when $\hat{I}_j \geq I_j$ and we have

$$f_j(\mathbf{q}_\pi) = f_j(\lambda_j \mathbf{p}_\pi) = \lambda_j f_j(\mathbf{p}_\pi) = I_j \ . \tag{25}$$

Case 2. $\lambda_j > 1$. This happens when $\hat{I}_j < I_j$ and we have

$$\begin{aligned} f_j(\mathbf{q}_\pi) &= \sqrt{\frac{1-\epsilon_j}{\epsilon_j}}\sqrt{\mathbf{q}_\pi^T \mathbf{R}_{\pi j}\mathbf{q}_\pi} + \sum_{\pi_m \in \boldsymbol{\pi} \backslash \boldsymbol{\pi}'} \overline{g}_{\pi_m \text{B}}^m q_{\pi_m \text{B}}^m + \hat{I}'_j \\ &\leq \lambda_j \left(\sqrt{\frac{1-\epsilon_j}{\epsilon_j}}\sqrt{\mathbf{p}_\pi^T \mathbf{R}_{\pi j}\mathbf{p}_\pi} + \sum_{\pi_m \in \boldsymbol{\pi} \backslash \boldsymbol{\pi}'} \overline{g}_{\pi_m \text{B}}^m p_{\pi_m \text{B}}^m \right) \\ &\quad + \hat{I}'_j \\ &= \lambda_j (f_j(\mathbf{p}_\pi) - \hat{I}'_j) + \hat{I}'_j \\ &= I_j \end{aligned}$$

Combining both cases, we see $f_j(\mathbf{q}_\pi) \leq I_j$ hold for each λ_j, and thus Property 1 is proved. ■

Property 1 means that constraints (11) are no longer our concern for feasibility. However, since the transmission powers can be scaled up, it is possible that the total transmission powers from a SU will exceed its device limit after the second scaling. In other words, constraints (10) may be violated and we need the third scaling.

Third scaling based on (10) To satisfy constraints (10), we will repeat the first scaling given by (18) and (19). A key observation is that the third scaling will not increase transmission powers $p_{\pi_m \text{B}}^m$'s and hence constraints (11) remain feasible. Further, constraints (12) only state the nonnegativity of $p_{\pi_m \text{B}}^m$'s, which holds trivially. Thus, we have reached a feasible solution from a starting point for the current subproblem and it is also a feasible solution for P2 and P1. Then we can calculate the objective value under this feasible solution.

By performing the above procedure for all the K_x subproblems (each with K_p starting points) in parallel, we have $K_x K_p$ feasible solutions (with their objectives values) for P2 and P1.

4.2.4 *Obtain the Final Solution*

Given the $K_x K_p$ feasible solutions, we will compare their objective values and the one with the highest objective value will be the final solution from GUC algorithm. As we shall show through experimental results in Section 6, this final solution achieves 90% optimality on average.

4.3 Summary of GUC algorithm design

A summary of GUC algorithm design is given in Fig. 3. The input includes network settings $P_{i\text{B}}^{\text{max}}$, I_j, ϵ_j; long-term

```
GUC:
1: Input: Network settings $P_{i\mathrm{B}}^{\max}$, $I_j$, $\epsilon_j$, $w_i$; long-term
   statistics $\overline{\mathbf{g}}_j$, $\mathbf{R}_j$; and short-term parameters $h_{i\mathrm{B}}^m$.
2: Output: A feasible solution with $x_{i\mathrm{B}}^m$ and $p_{i\mathrm{B}}^m$.
3: Generate $K_x$ subproblems
4:     Calculate $d_{i\mathrm{B}}^m$ for each corresponding $x_{i\mathrm{B}}^m$ in parallel
       based on (14).
5:     Find $D$ promising sets of $x_{i\mathrm{B}}^m$ and obtain the promis-
       ing space $\mathcal{S}$.
6:     Obtain $K_x$ samples based on the probabilities in (16)
       and fix $x_{i\mathrm{B}}^m$ correspondingly.
7: Solve the subproblems in parallel
8:     Obtain $K_p$ starting points of $p_{\pi_m \mathrm{B}}^m$ for each subprob-
       lem by (17).
9:     Apply the scaling-based local search for each start-
       ing point based on (19), (22) and (19).
10:    Calculate the objective value for each feasible solu-
       tion.
11: Obtain the Final Solution
12:    Compare the objective values from $K_x K_p$ solutions
       and find the one with the highest objective value.
```

Fig. 3. Summary of GUC algorithm design.

statistics $\overline{\mathbf{g}}_j, \mathbf{R}_j$; and short-term parameters w_i, $h_{i\mathrm{B}}^m$. The output is the best feasible solution with RB allocations $x_{i\mathrm{B}}^m$'s and transmission powers $p_{i\mathrm{B}}^m$'s.

Note that the information exchange of $P_{i\mathrm{B}}^{\max}$, I_j, ϵ_j, w_i is only done once during call setup time (or very rarely during the connection). Information exchange of $\overline{\mathbf{g}}_j$ and $\mathbf{R}_j$ is on the order of at least sub-seconds and hence does not pose any challenge. As for $h_{i\mathrm{B}}^m$, it can be updated as frequently as per TTI (125 μs), which is the same as CSI reporting as supported by current 5G standards. Thus, the overhead of information exchange in our design does not go beyond what is required in today's 5G standards.

Note that GUC is a fully centralized design (at BSs) since we believe centralized algorithms are more suitable for our problem than distributed algorithms. In particular, to be 5G-compliant, a centralized algorithm would fit more naturally than a distributed one. Further, in 5G, a centralized algorithm (such as GUC) that runs at pico BSs will enjoy more powerful computing resources and is not as energy-constrained as a distributed algorithm that runs on UEs. In terms of communication overhead, centralized algorithms only need information collection and results broadcasting. This overhead (both volume and required time) is much smaller and predictable than that in distributed algorithms [33].

5 GPU Implementation

In this section, we present our implementation efforts of GUC on NVIDIA GPU platforms. To meet our real-time requirement (125 μs), we need to efficiently allocate the available resources on GPU platforms in the implementation. In the rest of this section, we first offer an overview of GPU platform and describe key issues in implementation. Then we present the details of our implementation.

5.1 Overview of GPU and Key Issues

The most distinctive feature of GPU is its large number ($\sim 10^3$) of processing cores, which allows GPU to solve a large number of problems in parallel. GPU is also suitable for our problem due to its small size, low cost per floating point operations per second (FLOPS) [34, 35], and high flexibility of programming such as CUDA from NVIDIA [36]. Because of these advantages, we believe that GPU is the most suitable platform to implement GUC at a BS.

From an implementation perspective, our job is to make sure the GPU implements the operations of our GUC algorithm while minimizing its overall execution time. In GPU terminology, the basic unit to control the GPU operation is called a *Kernel* (a.k.a. Kernel function). We may have multiple kernels, which can be executed on GPU in arbitrary orders (sequential, parallel, or mixed) depending on the design. By defining the kernels and properly organizing their launching orders, we can implement the GUC algorithm.

The total time consumption on GPU consists of processing time and data access time from the memories. So the key issues in our implementations are: 1) How to reduce the processing time by optimizing thread allocations? 2) How to reduce data access time by proper memory management?

Threads allocations The minimum execution unit on GPU is called *thread*. A *thread block* is defined as a group of threads that share memory and computation resources. One thread block can have a maximum of 1024 threads. Since a kernel is executed as a grid of thread blocks, we need to decide how many thread blocks a kernel needs, how many threads for each thread block, and what are the processes of threads.

As for the processes of threads, we need to ensure the GUC algorithm is correctly implemented on GPU. Since threads operate in parallel, sometimes we need to pause certain threads until their required data has been processed by other threads and up-to-date. This can be done by thread synchronization. But thread synchronization slows down parallel computation and increases the overall execution time. Thus, we need to minimize the number of synchronizations as much as possible.

Memory management There are several types of memories on a GPU, each with a different performance. In our implementation, we use two types of memories in GPU called *global memory* and *shared memory.* Comparing to global memory, shared memory has a significantly shorter access time and can be broadcast to up-to 32 threads within one thread block. The downside of shared memory is its limited volume and no direct external access (i.e., cannot communicate directly with CPU memory). Thus, we need to carefully choose the memory type for each data in our implementation to reduce memory access time.

5.2 Implementation Details

Figure 4 shows a flow chart of our implementation, which includes five steps: two data transfers (between GPU and CPU) and three Kernels. Note that although the GUC algorithm also has three steps, the processes inside these three steps have been regrouped into three Kernels to minimize time consumption.

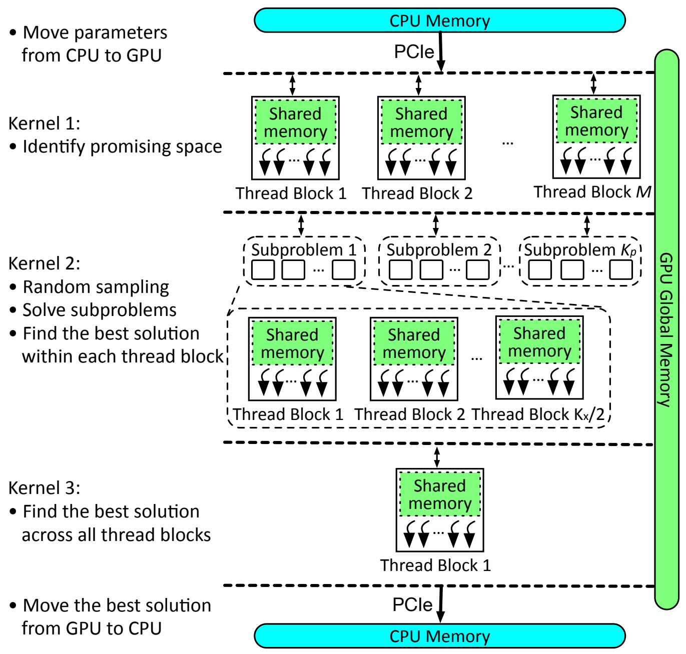


Fig. 4. Implementation of GUC algorithm on GPU

5.2.1 Move parameters from CPU to GPU

At the beginning of our implementation, we need to transfer data (i.e., network parameters, long-term settings, and short-term parameters) from CPU memory to GPU global memory. Note that only short-term parameters w_i and h_i need to be transferred to the GPU memory in each TTI while the long-term parameters $\mathbf{R}_j$ and $\overline{\mathbf{g}}_j$ can be stored in the GPU memory for use over TTIs until updated. Since we are using off-the-shelf GPU, such data transfer will go through a PCIe interface between CPU and GPU. To reduce this data transfer time, we use asynchronous data transfer between GPU and CPU, as recommended in [37]. Asynchronous data transfer means that the CPU will continuously issue commands to the GPU's command stream without waiting for the data transfer to be completed.

5.2.2 Kernel 1

The goal of Kernel 1 is to identify a promising space. For Kernel 1, we use M thread blocks and each one has N threads (i.e., MN threads in total). To identify a promising space, we need to calculate $d_{i\mathrm{B}}^m, i \in \mathcal{N}, m \in \mathcal{M}$. Since we have MN threads in total, each $d_{i\mathrm{B}}^m$ will be calculated by a thread in parallel. Then we find D promising SUs for each RB by finding the indexes of the max-D metrics regarding each RB, (i.e., find $\{s_1, s_2, \cdots, s_D\}$ from $\{d_{1\mathrm{B}}^m, d_{2\mathrm{B}}^m, \cdots, d_{N\mathrm{B}}^m\}$ such that $d_{s_1\mathrm{B}}^m \geq d_{s_2\mathrm{B}}^m \geq \cdots \geq d_{s_D\mathrm{B}}^m \geq$ others). Since the RB allocations on different RBs are independent, we use one thread block for an RB to perform these operations and this will give us a promising space $\mathcal{S}$ with D^M subproblems.

For each thread block, we need to find the max-D elements and their indexes from N given numbers. To reduce computation time through parallelization, we use D times of parallel reduction to obtain the max-D elements. Parallel reduction is widely used in GPU programming to reduce computation time, especially for operations such as finding the maximum (or minimum) and the sum of some given numbers [38]. In general, for operations involving n numbers, parallel reduction reduces the necessary iterations from $n-1$ to $\lceil log_2(n) \rceil$ using $n/2$ threads, which leads to a lower execution time. For instance, we need $\lceil log_2(N) \rceil$ iterations with $N/2$ threads to find the maximum among $\{d_{1\mathrm{B}}^m, d_{2\mathrm{B}}^m, \cdots, d_{N\mathrm{B}}^m\}$. Since execution time is our major concern and GPU has plenty of threads, we can use parallel reduction to reduce the execution time in our implementation.

As for memory management, we use shared memory for those frequently-used variables in Kernel 1, e.g., those temporary variables during parallel reduction, $d_{i\mathrm{B}}^m$'s, and the indexes of the promising RB allocations. The last two sets of variables will be copied to GPU global memory at the end of Kernel 1 for later steps since shared memory cannot be accessed externally. Note that data transfer between shared memory and GPU global memory can be done in parallel to reduce time consumption.

5.2.3 Kernel 2

Kernel 2 is the main component of our implementation, which has three parts: 1) random sampling K_x subproblems, 2) solving the subproblems, and 3) finding the best solution within each thread block. The reason why we combine the three parts into one Kernel is because of their close relationship, which avoids extra time for switching Kernels.

We use $K_x K_p/2$ thread blocks with each having $4M$ threads. That is $2MK_p$ threads for one subproblem and $2M$ threads for a starting point later on. We choose these parameters for three reasons. First, a design is considered "good" if it has a high "occupancy" [39], where occupancy is defined as the maximum utilization of each active Streaming Multiprocessor (SM) for a Kernel. Note that 100% "occupancy" is not always optimal since it is defined with *maximum* instead of average. Second, the number of threads in each thread block should not be too large. Otherwise, additional synchronizations would be needed in Kernel 2, which increases time consumption. Third, we use $2M$ threads to take advantage of GPU's parallelization where two independent calculations can be done simultaneously (such as the numerator and the denominator of a fraction).

In the following paragraphs, we will describe the three parts in detail.

Random sampling The goal of this part is to generate K_x subproblems. Since we only need K_x subproblems, the $K_x K_p/2$ thread blocks are equally divided into K_x groups and each group has $K_p/2$ thread blocks. Further, each group has the same sampling results that correspond to the same subproblem.

To randomly sample K_x subproblems, we need K_x feasible RB allocations inside the promising space. A feasible RB allocation is in the form of $\{\pi_1, \pi_2, \cdots, \pi_m\}$ where $\pi_m \in \{s_1^m, s_2^m, \cdots, s_D^m\}, m \in \mathcal{M}$. We use M threads to generate each subproblem and name these M threads from 0 to $M-1$. Then thread m will generate its allocated SU based on the following distribution

$$\begin{aligned}\mathbb{P}\{\pi_m = s_1\} &= d^m_{s_1\mathrm{B}}/d^m \ , \\ \mathbb{P}\{\pi_m = s_2\} &= d^m_{s_2\mathrm{B}}/d^m \ , \\ &\cdots \\ \mathbb{P}\{\pi_m = s_D\} &= d^m_{s_D\mathrm{B}}/d^m \ , \end{aligned} \tag{26}$$

where $d^m = d^m_{s_1\mathrm{B}} + d^m_{s_2\mathrm{B}} + \cdots + d^m_{s_D\mathrm{B}}$ is the normalization factor. The readers can easily verify that the random sampling per RB in (26) is equivalent to sampling the subproblems based on the distribution in (16).

Solve the subproblems In this part, we need to solve the K_x subproblems using our proposed method based on starting points and local search. Recall that in our GUC algorithm, each subproblem uses K_p starting points to generate K_p feasible solutions. The output of this part will be $K_x K_p$ feasible solutions and their corresponding objective values.

To solve a subproblem, we first generate the K_p starting points in parallel based on (17). A starting point is in the form of $\{p^1_{\pi_1\mathrm{B}}, p^2_{\pi_2\mathrm{B}}, \cdots, p^M_{\pi_M\mathrm{B}}\}$. Then we perform the scaling-based local search (i.e., (18)-(22)) to obtain a feasible solution from a specific starting point and calculate its objective value. After the scaling-based local search, we have $K_x K_p$ feasible solutions and we can use parallel reduction to calculate the corresponding $K_x K_p$ objective values.

There are many tricks to further reduce execution time on GPU. For instance, the calculations of $\hat{I}_j$ and $\hat{I}'_j$ can be done in parallel since we use $2M$ threads for one starting point. We can employ parallel reduction for the sums in (18)-(22). Here we discuss the calculation of $\mathbf{p}^T_\pi \mathbf{R}_{\pi j} \mathbf{p}_\pi$ in (20a) as an example to show how to further reduce the computation time of GUC.

Example 1. Calculation of $\mathbf{p}^T_\pi \mathbf{R}_{\pi j} \mathbf{p}_\pi$ in (20a).

Originally, we need to calculate M^2 polynomial terms corresponding to each element of $\mathbf{R}_{\pi j}$ for $\mathbf{p}^T_\pi \mathbf{R}_{\pi j} \mathbf{p}_\pi$. However, we can reduce the number of polynomial terms based on special structures of $\mathbf{R}_{\pi j}$. The original covariance $\mathbf{R}_j$ has the following properties:
(i) $\mathbf{R}_j$ is symmetric since it is the covariance of $\mathbf{g}_j$.
(ii) $\mathbf{R}_j$ is a band matrix meaning only several of its diagonals are non-zero. Notice that the values in $\mathbf{R}_j$ represent the correlation levels between RBs and it is well-known that the correlation of interference channel gains between two RBs decreases as they are further apart. To exploit this property, define L_j as the maximum subcarrier spacing that has correlations, meaning that an RB is correlated with at most $2L_j$ neighboring RBs. With the help of L_j, $\mathbf{R}_j$ is a band matrix with $(2L_j + 1)$ nonzero diagonals and typically L_j is substantially smaller than M. As a special case, $L_j = 0$ means that the interference channel gains $g^m_{\pi_m j}$'s are independent with each other and hence $\mathbf{R}_{\pi j}$ is a diagonal matrix.

Based on these two structures of $\mathbf{R}_i$, as its truncated version, $\mathbf{R}_{\pi j}$ is also a symmetric band matrix, given by

$$\mathbf{R}_{\pi j} = \begin{bmatrix} r_{11} & \cdots & r_{1(L_j+1)} & & & & \\ r_{21} & r_{22} & \cdots & r_{2(L_j+2)} & & & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \\ & & & & r_{(M-L_j)M} & \cdots & r_{MM} \end{bmatrix}$$

Clearly, there are $M(L_j + 1) - (L_j(1 + L_j)/2)$ non-zero elements in the upper triangle of $\mathbf{R}_{\pi j}$. Thus, we only need to calculate $M(L_j + 1) - (L_j(1 + L_j)/2)$ polynomial terms (compared to M^2 originally) for $\mathbf{p}^T_\pi \mathbf{R}_{\pi j} \mathbf{p}_\pi$, given as

$$\begin{aligned}\mathbf{p}^T_\pi \mathbf{R}_{\pi j} \mathbf{p}_\pi = &\sum_{m=1}^{M} (p^m_{\pi_m \mathrm{B}})^2 \cdot r_{mm} \\ &+ 2 \sum_{l_j=1}^{L_j} \sum_{m=1}^{M-l_j} p^m_{\pi_m \mathrm{B}} \cdot r_{m(m+l_j)} \cdot p^{m+l_j}_{\pi_{m+l_j}\mathrm{B}} \ . \end{aligned} \tag{27}$$

Another advantage of doing this is to reduce the memory needed to store $\mathbf{R}_j$ since we only need to store the non-zero elements in the upper triangle of $\mathbf{R}_j$. ■

Find the best solution within each thread block At this point, each thread block has multiple feasible solutions, we will find the one with the highest objective value within each thread block. Only these solutions will be copied to the GPU global memory for Kernel 3 and other solutions will be discarded. The benefit of doing this is to reduce the number of solutions to be copied to global memory and to be compared in Kernel 3, which saves execution time. In our design, we have $K_x K_p/2$ feasible solutions stored in GPU global memory after this step.

As for memory management in Kernel 2, we focus on shared memory. Specifically, we describe two typical types of data that should be stored in shared memory to reduce memory access time:
(i) data used only within a thread block. For example, $\hat{I}_j$ and $\hat{I}'_j$ in (20) are only used within a thread block for Kernel 2, and they are stored in shared memory.
(ii) data used outside of a thread block but also frequently accessed (read/write) by the threads within a thread block. For example, $p^m_{\pi_m \mathrm{B}}$'s need to be accessed in later steps but are frequently read and written by the threads within a thread block for Kernel 2. Therefore, $p^m_{\pi_m \mathrm{B}}$'s are stored in shared memory during the execution of Kernel 2 and will be copied to GPU global memory upon completion for later steps.

5.2.4 *Kernel 3*

The goal of Kernel 3 is to find the best solution across all thread blocks, which will be the final solution. We use one thread block for Kernel 3 to find the best solution by comparing the objective values among the $K_x K_p/2$ solutions. Clearly, this operation can be done through parallel reduction to reduce execution time. The best feasible solution (with the highest objective value) will be stored in GPU global memory.

5.2.5 *Move the best solution from GPU to CPU*

The last step of our implementation is to move the best solution from GPU global memory to CPU memory, which includes the feasible RB allocations $x^m_{i\mathrm{B}}$'s and transmit powers $p^m_{i\mathrm{B}}$'s.

6 Experimental Results

In this section, we evaluate the performance of our proposed GUC through experiments. All codes and data in our experiments are available at [40]. We will focus on the actual running time, threshold violation probability, and achieved objective values.

6.1 Settings

Throughout our experiments, we assume the distance between the macro BS and the pico BS is 400 meters and the radius of a picocell is 50 meters. We assume that the transmission power of macro BS on each RB is 46 dBm and each SU has a maximum transmission power of 20 dBm across all RBs [41]. The thermal noise on each RB is set to 1×10^{-7} mW.

For network topology, we consider fixed PUs and a varying number of SUs and RBs. We assume the pico BS is located at the origin and there are five fixed PUs ($J = 5$) near the picocell with coordinates $(60, 0), (-50, 25), (-40, 40), (35, -45)$, and $(55, 10)$. Though GUC can handle different interference thresholds and risk levels for the PUs, without loss of generality, we set the interference threshold for all PUs to $I_j = 3 \times 10^{-7}$ mW and use the same risk level ϵ for all PUs ranging from 0.01 to 0.5, i.e., $\epsilon_i = \epsilon \in [0.01, 0.5],\ i \in \mathcal{N}$.

The wireless channels are modeled by ITU path-loss [8] and Rayleigh fading. We consider two types of path-loss models as following:

1) The path-loss from the macro BS follows the ITU outdoor path-loss model as

$$PL(\text{dB}) = 128.1 + 37.6 \times \log_{10}(d_{\text{macro}}) \ , \tag{28}$$

where d_{macro} is the distance from the macro BS to the pico BS (in kilometers).

2) The path-loss from the SUs to the nearby PUs or the pico BS follows the ITU indoor path-loss model as

$$PL(\text{dB}) = 38 + 30 \times \log_{10}(d_{\text{pico}}) \ , \tag{29}$$

where d_{pico} is the distance from a SU to a nearby PU or the pico BS (in meters).

We consider channel gains can be either independent or correlated. In the correlated setting, we employ the well-known exponential correlation model [42] with a correlation factor $\rho = 0.7$, which represents an extreme case for correlated channel in a picocell. We set $L_j = 6$ under correlated channel, which means that two RBs are correlated if they are within 6 sub-carrier spacing and the correlation coefficients beyond 6 sub-carrier spacing are negligible (smaller than 0.01).

We emphasize that the channel models described here are only for us to generate the experimental parameters. Our GUC does not have any knowledge of these distributions. In other words, GUC is completely "blindfolded" with respect to these parameters in our experiments.

6.2 Case studies

In this subsection, we use fixed topologies as case studies to evaluate GUC's performance. We consider a typical picocell setting with 32 RBs and 20 SUs, i.e., $M = 32, N = 20$. We use three network topologies where the SUs are randomly distributed, clustering near the pico BS, and distributed around the picocell's edge.

6.2.1 Randomly distributed SUs

We first consider the case where SUs are randomly distributed in the picocell following a uniform distribution (see

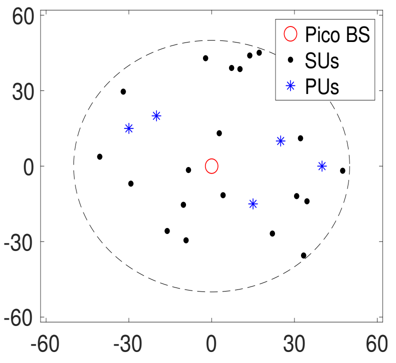


Fig. 5. A topology with SUs randomly distributed in the picocell.

Fig. 5). We use fixed weights w_i's for the SUs that are generated based on uniform distribution. w_i's are normalized such that the sum of w_i's among SUs is 1. For the parameter D used to identify the promising space, we choose $D = 2$ throughout our experiments, which means that the promising space has 2^M subproblems. As for the number of subproblems, we choose 128 subproblems for each run ($K_x = 128$) and use 24 different starting points ($K_p = 24$) for each subproblem with design parameter $\delta = 0.2$. This means that the final solution is the best solution selected from a total of 3072 ($K_x K_p$) feasible solutions.

We implement GUC on NVIDIA Telsa V1000 which has 7680 CUDA cores. As for CPU platform, we use Intel Xeon E5-2687w v4 and solve P2 in Gurobi 9.1 with 1% MIP gap. Since Gurobi cannot handle exponential cones from log functions (in the calculations of spectrum efficiency), we will upper bound each log term by a convex hull [43] and obtain a feasible solution and an upper bound on the optimal solution as well. This will reduce the computation time for solving the problem on CPU. Further, the obtained feasible solution is sufficiently close to the upper bound (from convex hull) and will serve as a benchmark for performance evaluation.

We perform 100 runs for each topology. For each run, we randomly generate transmission channel gain $h^m_{i\text{B}}$ based on the channel model described in Section 6.1. The results shown in this subsection are the average from these 100 runs.

Running time We first show the measured running time of GUC and Gurobi on CPU. The results are shown in Table 2 and we have two observations.

First, we see the average running time of GUC is smaller than the required 125 μs. In fact, we have checked the running time of each run and none of them exceeds 125 μs. This demonstrates that GUC meets the 5G timing requirement for a picocell scheduler. Comparing to the running time on CPU, GUC achieves at least 10^4 times of reduction in time under both independent and correlated channels.

Second, the running time under correlated channel is higher than the that under independent channel. This is as expected since under correlated channel, the problem itself requires more operations to solve, especially in calculating $\mathbf{p}_\pi^T \mathbf{R}_{\pi j} \mathbf{p}_\pi$.

Actual threshold violation probability Now we check whether or not the actual threshold violation probability is below the risk level, i.e., whether constraints (5) are satisfied

TABLE 2
Average running time (μs) for the network topology in Fig. 5

Implementation	Independent Channel	Correlated Channel
GUC on GPU	62.71	81.60
Gurobi on CPU	3.33×10^6	5.91×10^6

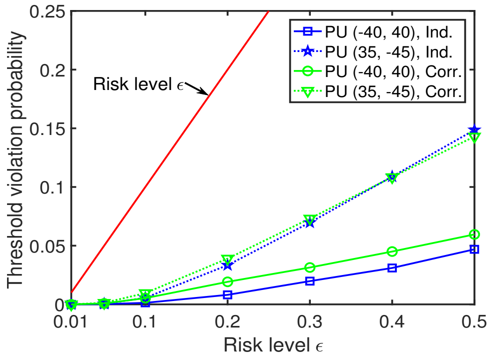


Fig. 6. PUs' threshold violation probabilities for the network topology in Fig. 5.

or not. To do this, after we obtain the RB allocations and transmission powers in each run, we generate 10,000 samples of interference channel gains g_{ij}^{m}'s. The actual threshold violation probability for each PU can be calculated by the portion of samples that the threshold is violated. For a fixed risk level ϵ, we show the average threshold violation probability from two PUs located at (-40,40) and (35,-45) since we found their violation probabilities are always greater than the other three PUs. The results are shown in Fig. 6.

As shown in Fig. 6, the actual threshold violation probabilities (on average) are indeed smaller than the risk level ϵ and they are different for the two PUs. In fact, we also checked the maximum threshold violation probabilities for all PUs among 100 runs and none of them exceeds the risk level ϵ, which means all nearby PUs are protected from GUC. Further, the actual threshold violation probabilities are increasing with a higher risk level ϵ, meaning that the SUs are taking advantage of the transmission opportunities provided by the PUs. The gap between the actual threshold violation probabilities and risk level ϵ affirms that our channel model (ITU pass loss and Rayleigh fading) is not the worst-case distribution, as discussed in [16]. Last but not the least, we see the correlation among channel gains only has a marginal impact on the actual threshold violation probabilities, which is expected since the correlation level inside a picocell is typically low (both in practice and in our considered setting).

Objective value The objective values obtained from GUC and the optimal solution from CPU are shown in Fig. 7. As shown in Fig. 7, the objective values are all increasing w.r.t. risk level ϵ. This is as expected since a higher risk level means more tolerance of interference threshold violation and thus SUs can increase their higher powers to achieve high throughput. Further, we see GUC achieves 90%∼92% optimality under independent and correlated channels compared with that from the optimal solution, which is excellent. Last, as shown in Fig. 7, the objective values under correlated channel are smaller than those under independent

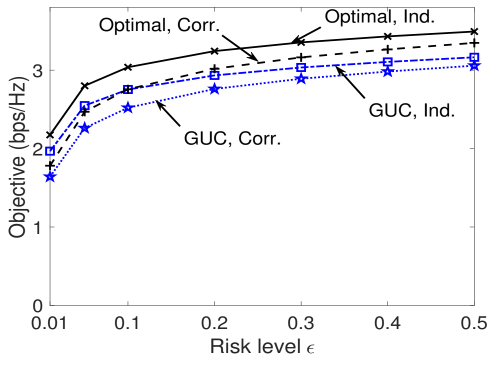


Fig. 7. Objective value for the network topology in Fig. 5

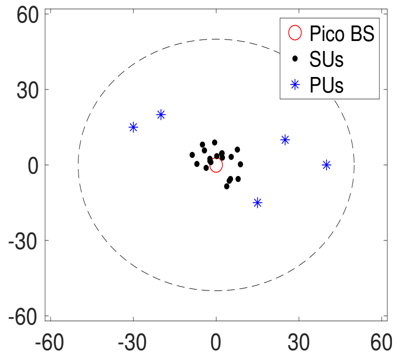


Fig. 8. A topology with all SUs clustered near the pico BS.

channel in both CPU and GUC solutions. In fact, such a loss due to correlation is 8% on average in both approaches. This is due to a smaller optimization space under correlated channel, which can be seen from constraints (9).

6.2.2 Two extreme topologies for SUs

We first consider a network topology where all SUs are clustering near the pico BS. This represents an extreme case for picocell settings. We randomly generate SUs inside the picocell based on uniform distribution while keeping the distance from each SU to the pico BS no greater than 10 meters. The network topology is shown in Fig. 8.

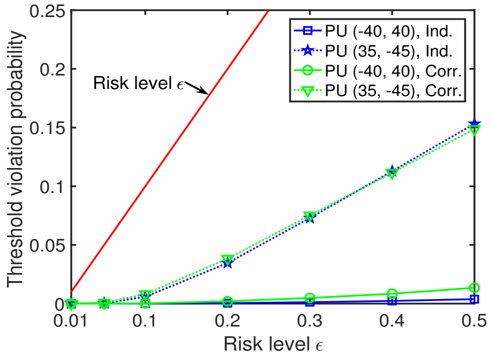
The measured running time is shown in Table 3 and we see the running time of GUC is smaller than the required 125 μs. Comparing to the running time on CPU, GUC achieves at least 10^4 and 10^5 times of reduction in running time under independent and correlated channels respectively.

The objective value and actual threshold violation probabilities for the two PUs are shown in Fig. 9. As shown in Fig. 9(b), the achieved spectrum efficiency is significantly higher than that from randomly distributed SUs. This is because the SUs are closer to the pico BS (thus high $h_{i\text{B}}^{m}$'s). Further, GUC achieves near-optimal performance (95% on average) compared to the optimal solution under independent and correlated channels.

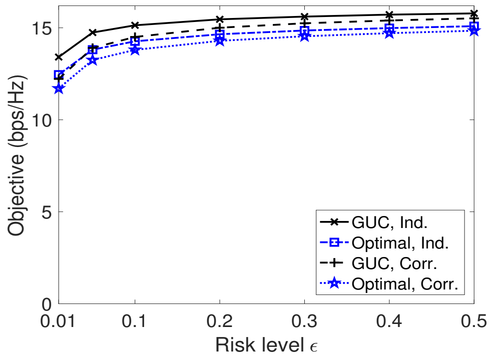
We next consider a network topology where all SUs are distributed around the picocell's edge. We randomly generate SUs inside the picocell based on uniform distribution but only keeping those locations to the pico BS within the 45∼50-meter ring. The topology is shown in Fig. 10. The measured running time is shown in Table 4. The objective

TABLE 3
Average running time (μs) for the network topology in Fig. 8

Implementation	Independent Channel	Correlated Channel
GUC on GPU	62.60	81.77
Gurobi on CPU	4.68×10^6	1.33×10^7



(a) Threshold violation probabilities for PUs



(b) Objective value

Fig. 9. Performance of GUC for the network topology in Fig. 8

value and actual threshold violation probabilities for the two selected PUs are shown in Fig. 11.

As shown in Table 4, the running time of GUC is smaller than the required 125 μs and is at least 10^4 smaller comparing to that from CPU. Further, comparing Table 2, Table 3, and Table 4, we see the running times of GUC are similar under the three netowork topologies while that from CPU varies widely. This is because the number of operations per thread from GUC is fixed under all three network topologies while the number of iterations in Gurobi can vary. Last, as shown in Fig. 11(b), the objective value is significantly lower than that from randomly distributed SUs since the SUs are further away from the pico BS (thus lower h_{iB}^{m}'s). We also see GUC achieves 90% and 86% optimality on average compared to the optimal solution obtained on CPU under independent and correlated channel respectively, which is satisfactory for practical purposes.

6.3 Varying RBs and SUs

6.3.1 Varying number of RBs

In this experiment, we evaluate GUC under a varying number of RBs. Specifically, we fix the number of SUs to $N = 20$ and set the number of RBs to $M = 32 \sim 80$.

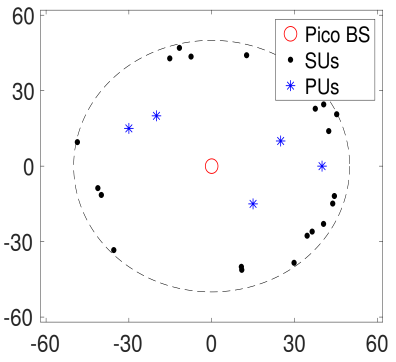


Fig. 10. A topology with all SUs distributed around the picocell's edge

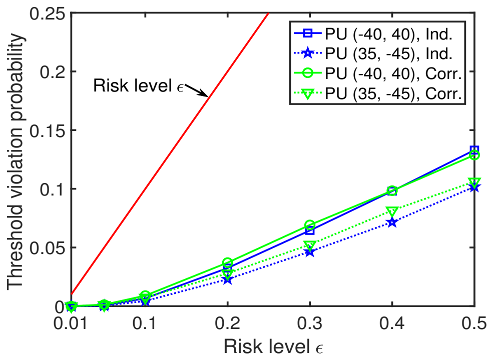
TABLE 4
Average running time (μs) for the network topology in Fig. 10

Implementation	Independent Channel	Correlated Channel
GUC on GPU	62.87	79.98
Gurobi on CPU	6.74×10^6	1.16×10^7

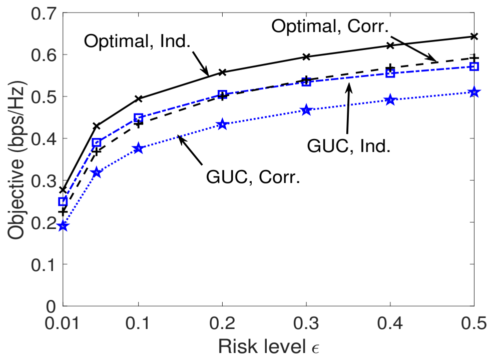
For each setting, we perform 100 runs. Since our main interest is running time, we will randomly generate a network topology, channel gains, and weights following uniform distribution in each run. Fig. 12 shows the average, maximum, and minimum of running time for each M (number of RBs).

As shown in Fig. 12, the running time of GUC increases monotonically w.r.t. M. This is because the complexity of Kernel 2 (the key component of GUC) largely depends on M, i.e., the number of decision variables in each subproblem. As M increases, the maximum number of operations for a thread also increases and we need to launch more threads for kernel functions. Though logically threads are launched totally in parallel but in fact, threads are launched in groups (typically 32 threads as a "warp") on physical processors. Thus, launching more threads could increase the overall time.

In Fig. 12, under independent channel, the running time of all instances meets the 125 μs requirement. But under correlated channel, the running time exceeds 125 μs when $M = 80$. This is due to the calculation of $\mathbf{p}_{\pi}^{T}\mathbf{R}_{\pi j}\mathbf{p}_{\pi}$ in (27), which involves M and $M(L_j+1)-(L_j(1+L_j)/2)$ polynomials terms under independent channel ($L_j = 0$) and correlated channel ($L_j = 6$) respectively. Note that we have K_xK_p subproblems and each subproblem has J terms of $\mathbf{p}_{\pi}^{T}\mathbf{R}_{\pi j}\mathbf{p}_{\pi}$. Thus, this slight difference in one term can lead to a significant difference in running time between independent and correlated channels. Nevertheless, such an issue can be relieved by choosing less subproblems (i.e., a smaller K_x), using less starting points per subproblem (i.e., a smaller K_p), or using customized GPU to reduce the data transfer time between GPU and CPU, which can be easily done by network operators. But in practice, $M \leq 64$ is expected for picocells since it only occupies a fraction of the macrocell's spectrum [29]. So we conclude that GUC will meet the 125 μs requirement under different M for practical purposes.

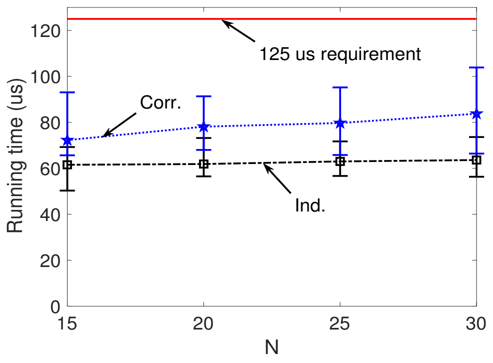


(a) Threshold violation probabilities for PUs

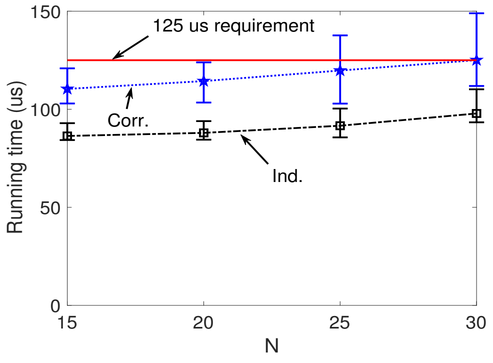


(b) Objective value

Fig. 11. Performance of GUC for the network topology in Fig. 10



(a) $M = 32$



(b) $M = 64$

Fig. 13. Running time of GUC under varying number of SUs

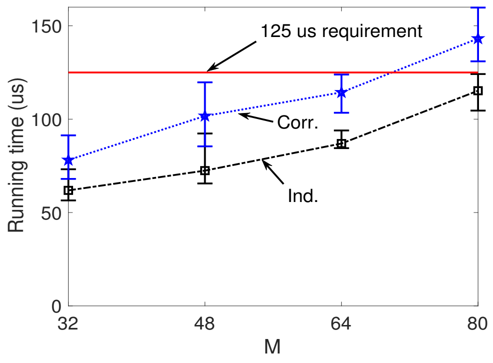


Fig. 12. Running time of GUC under varying number of RBs

6.3.2 *Varying number of SUs*

In this experiment, we evaluate GUC under a varying number of SUs. Specifically, we fix the number of RBs to $M = 32, 64$ and vary the number of SUs to $N = 15 \sim 30$. Similar with previous experiment, we perform 100 runs and in each run, we randomly generate a network topology, channel gains, and weights following a uniform distribution. Fig. 13 shows the average, maximum, and minimum of running time under varying N (number of SUs).

As shown in Fig. 13, both curves are relatively flat since the main time consumption of GUC is Kernel 2 whose complexity mainly depends on M. Further, when $M = 32$, the running time of GUC meets the 125 μs requirement in all experiments. However, when $M = 64$ and $N \geq 25$, the running time of GUC under correlated channel exceeds the 125 μs requirement, though these cases are rare for practical picocell settings. Nevertheless, similar with the previous case of $M > 64$, we can reduce the running time by using smaller K_x and K_p or a customized GPU for faster data transfer. In practice, $N \leq 25$ is expected for a picocell due to its small footprint, and thus we conclude that GUC meets the 125 μs requirement under different N for practical purposes.

7 CONCLUSIONS

In this paper, we studied underlay coexistence for 5G picocells sharing the same spectrum with a macrocell. Specifically, we perform scheduling and power control for the Secondary Users (SUs) to maximize the spectrum efficiency while protecting nearby Primary Users (PUs). To address channel uncertainty due to a lack of cooperation between PUs and SUs, we employed Chance-Constrained Programming (CCP) that allows occasion violations of PUs' interference thresholds. To meet the real-time requirement for a 5G picocell scheduler, we designed and implemented a novel scheduler called GPU-based Underlay Coexistence (GUC) that decomposes the original problem and solves subproblems in parallel on a large number of GPU cores. Through extensive experiments, we show that GUC meets the 125 μs timing requirement under typical picocell settings, which is at least 4 orders of magnitude reduction compared to conventional solution on CPU. Further, GUC achieves 90% optimality on average while guaranteeing threshold violation probabilities for the PUs. For future research, it is worth investigating the impact of channel uncertainty in more complicated scenarios, such as high user mobility, MU-MIMO, and nonorthogonal multiple access (NOMA).

REFERENCES

[1] S. Li, Y. Huang, C. Li, Y. T. Hou, W. Lou, B. Jalaian, S. Russell, and B. MacCall, "A real-time solution for underlay coexistence with channel uncertainty," in *Proc. IEEE GLOBECOM 2019*, pp. 1–6, Waikoloa, HI, Dec. 2019.

[2] A. Goldsmith, S. A. Jafar, I. Maric, and S. Srinivasa, "Breaking spectrum gridlock with cognitive radios: An information theoretic perspective," *Proceeds of the IEEE*, vol. 97, no. 5, pp. 894–914, May 2009.

[3] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, "What will 5G be?" *IEEE J. Selected Areas in Commun.*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[4] A. Ben-Tal and A. Nemirovski, "Robust solutions of uncertain linear programs," *Operations Research Letters*, vol. 25, no. 1, pp. 1–13, Aug. 1999.

[5] D. J. Costello and G. D. Forney, "Channel coding: the road to channel capacity," *Proceeds of the IEEE*, vol. 95, no. 6, pp. 1150–1177, June 2007.

[6] Y. Wang and Q.-F. Zhu, "Error control and concealment for video communication: a review," *Proceeds of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.

[7] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 451–515, Apr. 2000.

[8] 3GPP, *3GPP TR 36.931: Radio Frequency (RF) requirements for LTE Pico Node B*, June 2018, version 15.0.0. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2589.

[9] ——, *3GPP TR 36.932: Scenarios and requirements for small cell enhancements for E-UTRA and E-UTRAN*, July 2018, version 15.0.0. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2590.

[10] ——, *3GPP TR 38.802: Study on New Radio Access Technology Physical Layer Aspects*, Sep. 2017, version 14.2.0. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3066.

[11] ——, *3GPP TR 38.211: Physical channels and modulation*, March 2019, version 15.5.0. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213.

[12] Qualcomm. The 3GPP Release-15 5G NR design. Available: https://www.qualcomm.com/media/documents/files/the-3gpp-release-15-5g-nr-design.pdf (Last accessed: Feb. 2021).

[13] S. Wang, W. Shi, and C. Wang, "Energy-efficient resource management in OFDM-based cognitive radio networks under channel uncertainty," *IEEE Trans. on Commun.*, vol. 63, no. 9, pp. 3092–3102, Sept. 2015.

[14] N. Y. Soltani, S.-J. Kim, and G. B. Giannakis, "Chance-constrained optimization of OFDMA cognitive radio uplinks," *IEEE Trans. on Wireless Commun.*, vol. 12, no. 3, pp. 1098–1107, Mar. 2013.

[15] N. Mokari, S. Parsaeefard, P. Azmi, H. Saeedi, and E. Hossain, "Robust ergodic uplink resource allocation in underlay OFDMA cognitive radio networks," *IEEE Trans. on Mobile Computing*, vol. 15, no. 2, pp. 419–431, Feb. 2016.

[16] S. Li, Y. Huang, C. Li, B. A. Jalaian, Y. T. Hou, and W. Lou, "Coping uncertainty in coexistence via exploitation of interference threshold violation," in *Proc. ACM Mobihoc 2019*, pp. 7–80, Catania, Italy, July 2019.

[17] Y. Huang, S. Li, Y. T. Hou, and W. Lou, "GPF: A GPU-based design to achieve $\sim$100 μs scheduling for 5G NR," in *Proc. ACM MobiCom 2018*, pp. 207–222, New Delhi, India, July 2018.

[18] Y. Chen, Y. Huang, C. Li, T. Hou, and W. Lou, "Turbo-HB: A novel design and implementation to achieve ultra-fast hybrid beamforming," in *Proc. IEEE INFOCOM 2020*, pp. 1489–1498, Virtual Conference, May 2020.

[19] W. W.-L. Li, Y. J. Zhang, A. M.-C. So, and M. Z. Win, "Slow adaptive OFDMA systems through chance constrained programming," *IEEE Trans. on Signal Processing*, vol. 58, no. 7, pp. 3858–3869, July 2010.

[20] Z. Liu, Y. Xie, K. Y. Chan, K. Ma, and X. Guan, "Chance-constrained optimization in D2D-based vehicular communication network," *IEEE Trans. on Vehicular Technology*, vol. 68, no. 5, pp. 5045–5058, May 2019.

[21] T. A. Le, Q.-T. Vien, H. X. Nguyen, D. W. K. Ng, and R. Schober, "Robust chance-constrained optimization for power-efficient and secure SWIPT systems," *IEEE Trans. on Green Commun. and Networking*, vol. 1, no. 3, pp. 333–346, Sept. 2017.

[22] R. Atawia, H. Abou-Zeid, H. S. Hassanein, and A. Noureldin, "Joint chance-constrained predictive resource allocation for energy-efficient video streaming," *IEEE J. Selected Areas in Commun.*, vol. 34, no. 5, pp. 1389–1404, May 2016.

[23] C. Li, Y. Huang, Y. Chen, B. Jalaian, Y. T. Hou, and W. Lou, "Kronos: A 5G Scheduler for AoI minimization under dynamic channel conditions," in *Proc. IEEE ICDCS 2019*, pp. 1466–1475, Dallas, TX, May 2020.

[24] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Trans. on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 309–322, Feb. 2010.

[25] C. Jeon, K. Li, J. R. Cavallaro, and C. Studer, "Decentralized equalization with feedforward architectures for massive MU-MIMO," *IEEE Trans. on Signal Processing*, vol. 67, no. 17, pp. 4418–4432, Sept. 2019.

[26] H. Li, T. Zhang, R. Zhang, and X.-Y. Liu, "High-performance tensor decoder on GPUs for wireless camera networks in IoT," in *Proc. IEEE High Performance Computing and Communications (HPCC) 2019*, pp. 1619–1626, Zhangjiajie, China, Aug. 2019.

[27] A. Damnjanovic, J. Montojo, Y. Wei, T. Ji, T. Luo, M. Vajapeyam, T. Yoo, O. Song, and D. Malladi, "A survey on 3GPP heterogeneous networks," *IEEE Wireless Commun.*, vol. 18, no. 3, pp. 10–21, June 2011.

[28] V. Chandrasekhar, J. G. Andrews, and A. Gatherer, "Femtocell networks: A survey," *IEEE Commun. Magazine*, vol. 46, no. 9, pp. 59–67, 2008.

[29] F. Jin, R. Zhang, and L. Hanzo, "Fractional frequency reuse aided twin-layer femtocell networks: Analysis, design and optimization," *IEEE Trans. on Commun.*, vol. 61, no. 5, pp. 2074–2085, Sept. 2013.

[30] H.-B. Chang and I. Rubin, "Optimal downlink and uplink fractional frequency reuse in cellular wireless networks," *IEEE Trans. on Vehicular Technology*, vol. 65, no. 4, pp. 2295–2308, Apr. 2016.

[31] A. Furtado, L. Irio, R. Oliveira, L. Bernardo, and R. Dinis, "Spectrum sensing performance in cognitive radio networks with multiple primary users," *IEEE Trans. on Vehicular Technology*, vol. 65, no. 3, pp. 1564–1574, Mar. 2016.

[32] L. Wei and O. Tirkkonen, "Spectrum sensing in the presence of multiple primary users," *IEEE Trans. on Commun.*, vol. 60, no. 5, pp. 1268–1277, May 2012.

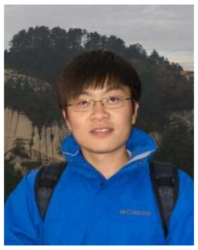
[33] P. Vamvakas, E. E. Tsiropoulou, and S. Papavassiliou, "Dynamic spectrum management in 5G wireless networks: A real-life modeling approach," in *Proc. IEEE INFOCOM 2019*, pp. 2134–2142, Paris, France, Apr. 29 - May 2, 2019.

[34] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in heterogeneous computing," *Scientific Programming*, vol. 18, no. 1, pp. 1–33, Jan. 2010.

[35] B. Betkaoui, D. B. Thomas, and W. Luk, "Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing," in *Proc. IEEE Field-*

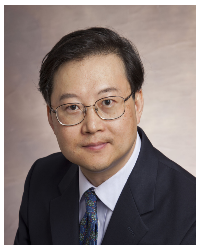
Programmable Technology (FPT) 2010, pp. 94–101, Beijing, China, Dec. 2010.
[36] N. Corp. Cuda toolkit. Available: https://developer.nvidia.com/CUDA-toolkit (Last accessed: Feb. 2021).
[37] M. Harris. How to overlap data transfers in CUDA C/C++. Available: https://devblogs.nvidia.com/how-overlap-data-transfers-cuda-cc/ (Last accessed: Feb. 2021).
[38] ——. Optimizing parallel reduction in CUDA. Available: https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf (Last accessed: Feb. 2021).
[39] ——. CUDA pro tip: Occupancy API simplifies launch configuration. Available: devblogs.nvidia.com/cuda-pro-tip-occupancy-api-simplifies-launch-configuration/ (Last accessed: Feb. 2021).
[40] S. Li. Experimental codes and data for GUC. Available: https://github.com/ShaoranLi/GUC.
[41] D. W. K. Ng, E. S. Lo, and R. Schober, "Energy-efficient resource allocation in OFDMA systems with large numbers of base station antennas," *IEEE Trans. on Wireless Commun.*, vol. 11, no. 9, pp. 3292–3304, Sept. 2012.
[42] R. K. Mallik, "On multivariate Rayleigh and exponential distributions," *IEEE Trans. on Information Theory*, vol. 49, no. 6, pp. 1499–1515, June 2003.
[43] Y. T. Hou, Y. Shi, and H. D. Sherali, *Applied Optimization Methods for Wireless Networks*, Chp. 5, pp. 110-112. Cambridge University Press, Apr. 2014.

Shaoran Li (S17) received the B.S. degree from Southeast University, Nanjing, China, in 2014 and the M.S. degree from the Beijing University of Posts and Telecommunications (BUPT), China, in 2017. He is a currently working toward his Ph.D. degree at Virginia Tech, Blacksburg, VA, USA. His research interests include algorithm design and implementation for wireless networks. In 2019, his paper won the Fred W. Ellersick MILCOM Award for the Best Paper in the unclassified technical program.



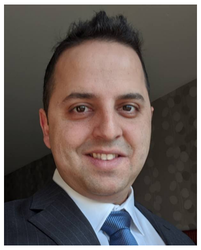
Yan Huang (M'15) is currently a senior system software engineer at the NVIDIA Corporation, Santa Clara, CA, USA. He received his Ph.D. degree in Electrical Engineering from Virginia Tech in 2020, and his B.S. and M.S. degrees in Electrical Engineering from Beijing University of Posts and Telecommunications in 2012 and 2015, respectively. During his Ph.D. study at Virginia Tech, he was awarded a Pratt Scholarship in 2019 and a Bindi Prasad Scholarship in 2020, respectively. He holds two U.S. and international patents. His research interests include GPU-accelerated real-time optimizations and machine learning for wireless communications and networking.

Chengzhang Li (S'17) is a Ph.D. student in the Bradley Department of Electrical and Computer Engineering at Virginia Tech, Blacksburg, VA, USA. He received his B.S. degree in Electronics Engineering from Tsinghua University, Beijing, China, in 2017. His current research interests are modeling, analysis and algorithm design for wireless networks, with a focus on Age of Information (AoI), 5G and ultra-low latency research.



Y. Thomas Hou (F'14) is Bradley Distinguished Professor of Electrical and Computer Engineering at Virginia Tech, Blacksburg, VA, USA, which he joined in 2002. His current research focuses on developing innovative solutions to complex science and engineering problems arising from wireless and mobile networks. He is also interested in wireless security. He has published over 300 papers in IEEE/ACM journals and conferences. His papers were recognized by nine best paper awards from IEEE and ACM. He holds six U.S. patents. He authored/co-authored two graduate textbooks: *Applied Optimization Methods for Wireless Networks* (Cambridge University Press, 2014) and *Cognitive Radio Communications and Networks: Principles and Practices* (Academic Press/Elsevier, 2009). Prof. Hou was named an IEEE Fellow for contributions to modeling and optimization of wireless networks. He was/is on the editorial boards of a number of IEEE and ACM transactions and journals. He was Steering Committee Chair of IEEE INFOCOM conference and was a member of the IEEE Communications Society Board of Governors. He was also a Distinguished Lecturer of the IEEE Communications Society.

Wenjing Lou (F'15) is the W. C. English Endowed Professor of Computer Science at Virginia Tech and a Fellow of the IEEE. Her research interests cover many topics in the cybersecurity field, with her current research interest focusing on wireless network security, trustworthy AI, blockchain, and security and privacy problems in the Internet of Things (IoT) systems. Prof. Lou is a highly cited researcher by the Web of Science Group. She received the Virginia Tech Alumni Award for Research Excellence in 2018. She received the INFOCOM Test-of-Time paper award in 2020. She was the TPC chair for IEEE INFOCOM 2019 and ACM WiSec 2020. She was the Steering Committee Chair for IEEE CNS conference from 2013 to 2020. She is currently a steering committee member of IEEE INFOCOM and IEEE Transactions on Mobile Computing. She served as a program director at the US National Science Foundation (NSF) from 2014 to 2017.



Brian A. Jalaian (M16) is currently a senior AI research scientist at Army Research Laboratory (ARL), AI Test & Evaluation Lead at DoD Joint Artificial Intelligence Center (JAIC), and an adjunct assistant professor in the Bradley Department of Electrical and Computer Engineering at Virginia Tech. He obtained his Ph.D. and MS degrees in Electrical Engineering in 2016 and 2013, respectively, and an MS in Industrial and Systems Engineering (Operation Research) in 2014, all from Virginia Tech, Blacksburg, VA. His research interests are optimization, AI safety and security, robust and resilient machine learning, and network science. He was a recipient of the 2019 Fred W. Ellersick MILCOM Award for the Best Paper in the unclassified technical program.

Stephen Russell is currently the Information Sciences Division Chief at the Army Research Laboratory. In addition to his research work, he has taught in the Department of Information Systems & Technology Management at George Washington University and instructed courses under the Federal CIO Certificate Program, Management, and Information Systems subject areas. Dr. Russell received a B.Sc. in Computer Science and M.S. and Ph.D. degrees in Information Systems from University of Maryland. He has published over 100 papers in his primary research areas of decision support systems, machine learning, systems architectures, and intelligent systems. He has several years of information technology experience in the telecommunications, healthcare, and manufacturing industries. Dr. Russell has also been a serial entrepreneur, owning companies that have specialized in software engineering, information resource management services, and telecommunications equipment manufacturing. Currently, his research is focused on Internet of Things and its applicability to the U.S. Army and the U.S. Military. He leads the ARL research program on Internet of Battlefield Things (IOBT), which is focused on multiple challenges of incorporating IoT ideas and capabilities within the battlefield environment.