Optimal Scheduling and Allocation for IC Design Management and Cost Reduction

PRABHAV AGRAWAL, UC San Diego
MIKE BROXTERMAN, Qualcomm, Inc.
BISWADEEP CHATTERJEE, Qualcomm India Private Limited
PATRICK CUEVAS and KATHY H. HAYASHI, Qualcomm, Inc.
ANDREW B. KAHNG, PRANAY K. MYANA, and SIDDHARTHA NATH, UC San Diego

A large semiconductor product company spends hundreds of millions of dollars each year on design infrastructure to meet tapeout schedules for multiple concurrent projects. Resources (servers, electronic design automation tool licenses, engineers, and so on) are limited and must be shared – and the cost per day of schedule slip can be enormous. *Co-constraints* between resource types (e.g., one license per every two cores (threads)) and dedicated versus shareable resource pools make scheduling and allocation hard. In this article, we formulate two mixed integer-linear programs for optimal multi-project, multi-resource allocation with task precedence and resource co-constraints. Application to a real-world three-project scheduling problem extracted from a leading-edge design center of anonymized Company X shows substantial compute and license costs savings. Compared to the product company, our solution shows that the makespan of schedule of all projects can be reduced by seven days, which not only saves ∼2.7% of annual labor and infrastructure costs but also enhances market competitiveness. We also demonstrate the capability of scheduling over two dozen chip development projects at the design center level, subject to resource and datacenter capacity limits as well as per-project penalty functions for schedule slips. The design center ended up purchasing 600 additional servers, whereas our solution demonstrates that the schedule can be met without having to purchase any additional servers. Application to a four-project scheduling problem extracted from a leading-edge design center in a non-US location shows availability of up to ∼37% headcount reduction during a half-year schedule for just one type of chip design activity.

Categories and Subject Descriptors: B.7.2 [**Design Aids**]: Design Schedule and Cost Optimization

General Terms: Design, Optimization

Additional Key Words and Phrases: Design cost optimization, resource scheduling, project scheduling

ACM Reference Format:
Prabhav Agrawal, Mike Broxterman, Biswadeep Chatterjee, Patrick Cuevas, Kathy H. Hayashi, Andrew B. Kahng, Pranay K. Myana, and Siddhartha Nath. 2017. Optimal scheduling and allocation for IC design management and cost reduction. ACM Trans. Des. Autom. Electron. Syst. 22, 4, Article 60 (June 2017), 30 pages.
DOI: http://dx.doi.org/10.1145/3035483

Authors' addresses: P. Agrawal, A. B. Kahng, P. K. Myana, and S. Nath, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093; M. Broxterman, P. Cuevas, and K. H. Hayashi, Qualcomm Inc., 5775 Morehouse Drive, San Diego, CA 92121; B. Chatterjee, Qualcomm Technology India Pvt. Ltd., Plot 153-154, EPIP, Phase II, Whitefield, Bangalore KA 560066.

DOI: http://dx.doi.org/10.1145/3035483

1. INTRODUCTION

Since 2001, the *International Technology Roadmap for Semiconductors* [ITRS] chapter on Design Technology has presented a Design Cost model calibrated to mobile system-on-chip (SOC) products (e.g., Qualcomm Snapdragon [Snapdragon] and Samsung Exynos [Exynos]) that are the main processing cores of tablets and smartphones and their associated development costs [Chan et al. 2014; Kahng and Smith 2002; Smith 2014]. For well over a decade, the Design Cost model has documented design costs of tens of millions of dollars for a single SOC product. Major contributors to design cost include engineering headcount, compute infrastructure (servers, filers, datacenters), and electronic design automation (EDA) tool licenses. The large investment requirement for new product development stifles semiconductor startup activity and innovation, and has arguably contributed to consolidation and a slowdown of growth for the industry.

Today, a large semiconductor product company will spend hundreds of millions of dollars annually on design infrastructure (e.g., datacenters, EDA tools, design teams) to meet tapeout schedules for multiple concurrent projects. Resources (e.g., servers, licenses, engineers) are limited and must be shared across projects. Not only are schedule slips extremely costly but, as highlighted in recent years (e.g., the "How Green is My Silicon Valley" plenary panel at the 2009 Design Automation Conference (DAC) [DAC09]), there is now tremendous concern to reduce the energy footprint of semiconductor integrated circuit (IC) design. In contrast to traditional scheduling optimizations seen in the operations research and industrial engineering literature, IC design flows often exhibit *co-constraints* between resource types (e.g., one license needed per every two cores used in a multi-threaded tool run[1]). Common design center practices, such as the setting up of dedicated vs. shareable resource pools as permitted by LSF-type gridware [LSF], also make scheduling and allocation hard. Further, design managers, while increasingly able to track and diagnose design activity [Fenstermaker et al. 2000; RTDA], have no decision support tools to help determine the resource investments (e.g., is it better to add 500 more servers or 50 more timing analysis tool licenses?) that enable schedule requirements to be met with minimum cost. Thus, a company may leave millions of dollars and gigawatt-hours per year – as well as weeks of schedule time – on the table. In a competitive and cost-driven industry, there is an urgent need to recover such wasted resources.

In the field of operations research, Kolisch and Hartmann [1999], Kolisch et al. [1992], and Kolisch and Sprecher [1996] give an integer-linear programming (ILP) formulation to solve the *resource-constrained project scheduling problem* (RCPSP). The formulation optimally allocates renewable, non-renewable, and doubly-constrained resources across multiple activities (with precedence constraints) in a project. The objective of the formulation is to minimize the makespan of a project with multiple activities. We extend this formulation in the context of IC design cost optimization in various ways. Specifically, we describe two mixed integer-linear programming (MILP) formulations that efficiently and optimally perform multi-project, multi-resource allocation with complex task precedence and resource co-constraints. The first is the *Schedule Cost Minimization* (SCM) formulation, and the second is the *Resource Cost Minimization* (RCM) formulation. We solve these two general resource-constrained project scheduling problems that arise in a multi-tenanted, heterogeneous, *high-throughput computing* (HTC) environment. A problem instance consists of projects that can be scheduled

[1]Maintaining design schedules with constant engineering headcount, even as SOC complexities continue to scale, increasingly relies on multithreading (e.g., detailed routing, static timing analysis, physical verification) and/or massively distributed tool runs (e.g., to perform functional verification).

in parallel, each involving multiple activities, in which each activity must consume prescribed amounts of resources to reach completion. The goal is to schedule the projects either with minimum total loss according to given penalty functions or with minimum number of resources consumed per time unit.[2] In Section 4.1, we describe an instance with three projects, 11 activities with various precedence constraints per project, and five types of resources that must be completed within 90 days. Our MILP formulation captures such types of problems in a straightforward manner, as we demonstrate in Section 4.1.

The challenge in practice for a large semiconductor design organization is to provide just-in-time resources for each project, such that (i) project execution is not delayed by resource starvation and (ii) utilization of each resource type satisfies resource limits or usage policies. Current industry dynamics lead to strict boundary conditions (e.g., time-to-market, tapeout deadline), and constrained capital spending pushes business units to seek increased productivity through maximum utilization of existing resources. Today, resource planning and allocation, especially involving allocation of multiple disparate resource types, has largely been dictated by heuristics and historical experience. Decision support is urgently needed for "course corrections" and understanding of the impact of resource allocation decisions. With this as background, our main contributions are as follows.

(1) We model two resource-constrained optimal project scheduling formulations, SCM and RCM, as MILPs. Our formulations handle multiple projects, multiple activities with precedence constraints, and multiple types of resources.[3]
(2) We handle *co-constraints* between resource types and allocation of resources from multiple (fully-shared, conditionally-shared, segregated) resource pools. Each pool may have a different penalty function, capturing real-world scenarios in a large SOC design company. To our knowledge, we are the first to consider co-constraints between resource types.
(3) We optionally enforce *stability* constraints that upper-bound the change in a project's allocated resources between successive timesteps.
(4) Application of SCM to a three-project scheduling problem extracted from a leading-edge design center of Company X[4] shows substantial compute and license cost savings compared to the actual allocation/scheduling solution used by the product company. Our solution reduces the schedule makespan of all projects by 1.4 work-weeks,[5] i.e., ∼2.7% of annual design infrastructure cost. (Per "Moore's Law," the semiconductor industry advances at ∼1% in a calendar week [Moore's Law]. Therefore, during this time, the semiconductor industry advances by more than 1%.) We also demonstrate the scheduling of two dozen chip development projects at the design center level, subject to resource and datacenter capacity limits as well as per-project penalty functions for schedule slips. The design center was unable to solve this problem and ended up purchasing 600 additional servers to avoid schedule

[2]A typical real-world HTC environment has multiple concurrent projects – each working on a specific schedule that is largely non-negotiable and each having different workload characteristics in terms of infrastructure requirements.
[3]Note that we do not solve arbitrary-sized MILP formulations in this work. The context of our formulation and ranges of our inputs pertain to IC design projects; very few, if any, semiconductor companies in the real world would face problem instances with complexities larger than what we study in the article. Further, we believe that on the time scales of SOC implementation, even a couple of days of runtime is tolerable if the return is weeks of schedule gain or millions of dollars of design cost reductions.
[4]Owing to confidentiality reasons, we cannot reveal the name of the company; we refer to it as Company X.
[5]In the semiconductor industry, we typically refer to one "work-week" as five working days in a week.

Table I. Representative Previous Works

Reference	Formulation	Objective	Modes	Preemptive	Resource Co-constraints	Conditionally-Shared Resources
Ayala and Artigues [2010]	ILP	throughput	✗	✗	✗	✗
Baptiste and Demassey [2004]	LP	makespan	✗	✗	✗	✗
Bienstock and Zuckerberg [2009]	LP	cost	✓	✓	✗	✗
Bonfietti et al. [2014]	CP	throughput	✗	✓	✗	✗
Christofides et al. [1987]	ILP, LP	cost	✗	✓	✗	✗
Keller and Bayraksan [2009]	Stochastic ILP	cost	✗	✗	✗	✗
Kolisch et al. [1992], Kolisch and Sprecher [1996]	ILP	makespan	✓	✗	✗	✗
Kramer and Hwang [1991]	MILP, LP	cost	✓	✓	✗	✗
Li et al. [2009]	MILP	makespan	✗	✓	✗	✗
Mohanty and Nayak [2011]	PSO	cost	✗	✓	✗	✗
Qiong et al. [2010]	ACO	makespan	✗	✗	✗	✗
Salewski et al. [1997]	ILP	cost	✓	✗	✗	✗
Our work	MILP	cost/makespan	✗	✓	✓	✓

slips. Our solution shows that the schedule requirements could have been met without purchasing any additional servers.

(5) Application of RCM to a four-project scheduling problem extracted from a leading-edge design center of Company X shows substantial human resource costs left on the table by the actual allocation/scheduling solution used by the company. For a particular activity related to chip design, our solution reduces headcount by 37%, which translates to ∼$3.7 million in savings at that particular (non-US) design center. Our solver can also provide decision support via "what-if" analyses of cost and schedule trade-offs.

(6) Of separate interest is the description of our test-case generator that we use to perform scalability and sensitivity studies. We propose to make our generator and solvers open-source, as prototyped at MILP-Solver.

The remainder of this article is organized as follows. Section 2 reviews relevant prior work. Section 3 describes our MILP formulations. In Section 4, we describe experimental validation of benefits from our MILP formulations, using three instances from a worldwide top-5 semiconductor company. We present our conclusions and outline future work in Section 5.

2. RELATED WORK

Resource-constrained project scheduling has been solved in many different settings with varying constraints and/or objective functions. Table I places our present work in the context of representative previous works on resource scheduling with multiple activities. A common objective is to minimize the makespan [Baptiste and Demassey 2004]. Objective functions studied typically minimize project cost given time-dependent and/or resource-dependent penalties [Kramer and Hwang 1991; Talbot 1982].

Several previous works solve the scheduling problem for a single project with multiple activities [Bienstock and Zuckerberg 2009; Christofides et al. 1987; Keller and Bayraksan 2009; Kolisch and Sprecher 1996; Mohring et al. 2001; Salewski et al. 1997]. The activities can be either preemptive or nonpreemptive [Baptiste and Demassey 2004; Kolisch and Sprecher 1996; Salewski et al. 1997; Talbot 1982]. Kolisch and Sprecher [1996] and Kolisch et al. [1992] formulate the RCPSP and propose methods to generate RCPSP instances. They present the PSPLIB and MPSPLIB benchmark

suites, along with optimal as well as heuristic solutions.[6] Further variations involve the scheduling of activities that can execute in multiple *modes* [Bienstock and Zuckerberg 2009; Kolisch and Sprecher 1996; Kramer and Hwang 1991; Salewski et al. 1997; Talbot 1982]. These works consider that the resource usage and the time taken by an activity can vary across available modes; they provide optimal scheduling solutions across combinations of modes of activities. Mohring et al. [2001] and Christofides et al. [1987] provide branch-and-bound algorithms to solve the resource-constrained multi-activity single project scheduling problem. Mohring et al. [2001] further try to identify special cases that are solvable in polynomial time. Generally, solution frameworks involve linear or integer linear programming, although stochastic [Keller and Bayraksan 2009] and nonlinear [Bonfietti et al. 2014] formulations have also been studied. Cyclic scheduling has been addressed in Ayala and Artigues [2010] and Bonfietti et al. [2014], in which sets of activities are executed indefinitely over time in a periodic fashion. The work of Keller and Bayraksan [2009] is noteworthy in that its formulation permits temporary resource expansion, albeit for a penalty that features in the objective function. This has some similarity to our formulation presented later, which has different penalties for resources used from different resource pools. (The formulation provided in Keller and Bayraksan [2009] does not include precedence constraints within activities or a number of other aspects of our formulation.)

To optimize human resources at an enterprise scale, Li et al. [2009] minimize the makespan of a single project with multiple activities, subject to upper bounds of human resources. Mohanty and Nayak [2011] propose a particle swarm optimization (PSO) algorithm to optimize the trade-off between cost and profit when a given number of employees are assigned to an activity. Their formulation considers employees with different skill competencies for different activities. Qiong et al. [2010] propose an ant colony optimization (ACO) algorithm to minimize the makespan of a project with multiple activities and precedence constraints between the activities. The activities are assumed to be non-preemptive, and the algorithm is applicable to general parallel machine-scheduling problems.

Several commercial tools and services exist today [Dassult Systems; IC Manage; inMotion; Nefelus; Salesforce] that serve design project management needs. Some of these tools are specific to IC design [Dassult Systems; IC Manage; Nefelus], whereas the other tools can serve project management needs for any industry. Our work is not comparable to these tools because our work is combinatorial optimization-based and solves formulations that, to our knowledge, are not addressed by any commercial product. We have experimented with multiple tools for forecasting and performing what-if analyses. However, none of these commercial tools are flexible to enable analyses in different scenarios that large design companies work on, which has led to the development of custom tools and methods for project planning. Today, large design companies use a mix of in-house customer methods, statistical packages, business reporting tools and large-scale production databases for project planning.

Comparisons to works on datacenter job allocations. Our work uses an objective function similar to that seen in works from the datacenter literature that propose algorithms to handle job scheduling within a datacenter, e.g., to minimize the makespan as well as other penalty functions. With energy consumption a major concern in modern datacenters, recent formulations by Friese et al. [2012] propose multi-objective optimization of makespan and energy consumption. However, formulations for datacenters are focused on providing job scheduling solutions either in real-time or "online"

[6]We have compared optimal solutions from our formulation with the optimal solutions of benchmarks from PSPLIB in Section 4.

[Li et al. 2014], often by using live data from various thermal, network, and rack utilization sensors. By contrast, our optimization is performed "offline," that is, we do not monitor status of project executions in real-time during the optimization of our objective function.

Other distinctions from previous works. While a number of previous works address optimizations related to resource-constrained project scheduling, they cannot address important use cases that arise for large SOC product companies. Our formulations address real-world use cases that incorporate the following: (i) resource co-constraints, (ii) tethering forecast resource allocations, and (iii) simultaneous allocation of three different categories of resources (*Fully-shared*, *Segregated*, and *Conditionally-shared*). Our formulations also handle *stability* constraints so that allocation of resources (in particular, engineers) are shuffled as infrequently as possible across projects. This induces a trade-off between schedule cost and frequency of task switching. Overall, we enable management to identify the minimum cost (in terms of any penalty functions deemed appropriate for the situation) of project completion within a set period of time, capturing many constraint types that arise in the industry. Our solver can also help analyze how varying resource allocation affects cost and schedule of product tapeout. We demonstrate a use case of handling late-breaking bugs in one project without major disruptions in allocations of other projects.

3. PROBLEM FORMULATIONS

We now present (i) notations used in our discussion, (ii) resource categories that arise in multi-tapeout project scheduling, and (iii) our MILP formulations. We have spent considerable time working with technical management at one of a worldwide top-5 semiconductor company's design centers to arrive at the optimization formulations described later. Table II gives notations used in our work. "I" represents an input to the MILP and "O" an optimization variable. We also indicate which notations are used in each of the SCM and RCM formulations.

3.1. Resource Pool Types

Chip design companies typically have three pools for each resource type. Resource types include compute nodes, memory, storage, and people [Qualcomm personal communication].

Fully-shared resources are shared across all projects. We use $r_{i,j,k,t}$ to denote the number of fully shared resources of type k used by activity $a(i, j)$ of project P_i at time t. For example, if there are two projects P_1 and P_2 with one activity each and 20 fully-shared resources of type k are available, then P_1 and P_2 can share these 20 resources among themselves such that $r_{1,1,k,t} + r_{2,1,k,t} \leq 20$.

Segregated/dedicated resources are allocated exclusively to a specific project. These resources are not available for use by any other projects at any time. We use $q_{i,j,k,t}$ to denote the number of segregated resources of type k used by activity $a(i, j)$ of project P_i, at time t. For example, if there are two projects P_1 and P_2 with one activity each and they are respectively allocated 10 and 20 segregated resources of type k, then $q_{1,1,k,t} \leq 10$, and $q_{2,1,k,t} \leq 20$.

Conditionally-shared resources are allocated to each project, but any resource unused by a project may be used by other projects. We use $y_{i,j,k,t}$ to denote the number of conditionally-shared resources of type k used by activity $a(i, j)$ of project P_i at time t. For example, if there are two projects P_1 and P_2 with one activity each and they are respectively allocated 10 and 20 conditionally-shared resources of type k, then $y_{1,1,k,t} \leq 10$ and $y_{2,1,k,t} \leq 20$. We use the notation $z_{i,j,k,t}$ to denote the number of resources of type k used

Table II. Notations Used in Our Work

Parameter	Description	I/O	Formulation
N	Total number of projects	I	SCM, RCM
T	Maximum duration over all projects; $t = 1, 2, \ldots, T$	I	SCM, RCM
P_i	Projects indexed by $i = 1, 2, \ldots, N$	–	SCM, RCM
$J(i)$	Total number of activities for P_i	I	SCM, RCM
$a_{i,j}$	P_i's activities, where $j = 1, 2, \ldots, J(i)$	–	SCM, RCM
$H^a(i, j)$	Set of predecessor activities of $a_{i,j}$ that must complete before $a_{i,j}$ can start	I	SCM, RCM
K	Available resource types	I	SCM, RCM
$R_{k,t}$	Upper bound (UB) on # resources of type k at time t	I	SCM
$H^r(i, j, k)$	Set of predecessor resources for resource type k for $a_{i,j}$	I	SCM
$g(i, j, h, k, t)$	Function that sets a UB on # resource type k at any time t, for each predecessor $h \in H^r(i, j, k)$	I	SCM
$L_{i,j,k}$	# resources of type k required to complete $a_{i,j}$	I	SCM, RCM
$U_{k,t}$	UB on # fully-shared resources of type k at time t	I	SCM
$\tilde{U}_k$	UB on # fully-shared resources of type k at any time t	O	RCM
$V_{i,k,t}$	UB on # segregated resources of type k for P_i at time t	I	SCM
$M_{i,k,t}$	UB on # conditionally-shared resources of type k for P_i at time t	I	SCM
$G_{i,k,t}$	UB on total # resources of type k used by P_i at time t	I	SCM, RCM
$B_{i,k,t}$	UB on change in resources consumed by P_i from $t-1$ to t	I	SCM, RCM
$d_{i,j}^{nom}$ (e_i^{nom})	Nominal duration of $a_{i,j}$ (P_i)	I	SCM, RCM
$C_{i,j}^a(t)$ ($C_i^p(t)$)	Penalty function for $a_{i,j}$ (P_i) at time t	I	SCM, RCM
C_k	Weight for resource type k	I	RCM
C	Cost of switching activities/projects	I	SCM+
$w_{i,j,k,t}$	# resources of type k consumed by $a_{i,j}$ at time t, given by forecast resource allocation	I	SCM
δ	% of variation allowed in $w_{i,j,k,t}$	I	SCM
$s_{i,j}^{nom}$ ($f_{i,j}^{nom}$)	Nominal start (finish) time of $a_{i,j}$ for tethering constraints	I	SCM, RCM
$r_{i,j,k,t}$	# fully-shared resources of type k consumed by $a_{i,j}$ at time t	O	SCM, RCM
$q_{i,j,k,t}$	# segregated resources of type k consumed by $a_{i,j}$ at time t	O	SCM
$y_{i,j,k,t}$	# conditionally-shared resources of type k consumed by $a_{i,j}$ at time t	O	SCM
$z_{i,j,k,t}$	# unused conditionally-shared resources of type k consumed by $a_{i,j}$ at time t	O	SCM
$s_{i,j}$ ($f_{i,j}$)	Start (finish) time of $a_{i,j}$	O	SCM, RCM
$S_{i,j,t}$ ($F_{i,j,t}$)	0-1 variable, set to 1 if $t \geq s_{i,j}$ ($t \geq f_{i,j}$); 0 otherwise	O	SCM, RCM

by activity $a(i, j)$ of project P_i at time t from the pool of unused conditionally-shared resources of other projects. In the preceding example, we have $z_{1,1,k,t} \leq 20 - y_{2,1,k,t}$ and $z_{2,1,k,t} \leq 10 - y_{1,1,k,t}$.

Figure 1 illustrates two scenarios with three projects: A, B, and C. Each project has one activity and consumes resource type k at time t. Each project may use resources from any of the three pools with the following constraints: (i) segregated resources $q_{i,j,k,t}$ consumed by a project cannot exceed the upper bound $V_{i,k,t}$, as shown in Figure 1, and (ii) conditionally-shared resources $y_{i,j,k,t}$ consumed by a project cannot exceed $M_{i,k,t}$. Figure 1(a) shows a feasible allocation of resources from each pool. Projects A and B have a total of eight units of unused resources in their conditionally-shared pools after allocation of resources from each pool.[7] Project C uses five out of these eight units, i.e., $z_{C,j,k,t} = 5$. The total number of fully-shared resources consumed by all three projects,

[7]Projects A, B, and C use 4, 4, and 5 resources from their respective segregated pools, which are within the upper bounds $V_{A,k,t} = V_{B,k,t} = V_{C,k,t} = 5$.

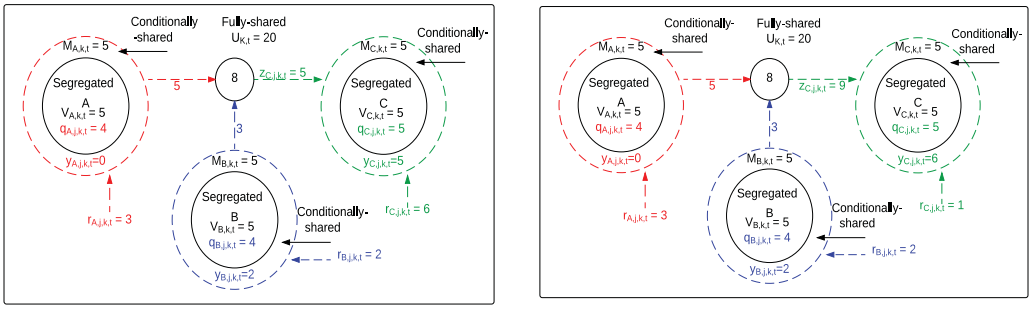


Fig. 1. Examples showing (a) feasible and (b) infeasible allocations of resources among three projects, A, B, and C.

i.e., $3+2+6=11$, cannot exceed $U_{k,t}=20$. Figure 1(b) shows an allocation of resources that is infeasible because $y_{C,j,k,t}=6>M_{C,k,t}=5$. Furthermore, project C uses more resources from the unused conditionally-shared resource pool label, i.e., $z_{C,j,k,t}>8$.

3.2. MILP Description of the Schedule Cost Minimization Formulation

Given the inputs listed in Table II for the SCM formulation, we seek to minimize the total cost (i.e., sum of schedule penalties) of all projects:

$$\text{minimize} \sum_{i=1}^{N}\sum_{t=1}^{T} C_i^p(t) + \sum_{i=1}^{N}\sum_{j=1}^{J(i)}\sum_{t=1}^{T} C_{i,j}^a(t) \tag{1}$$

This optimization is subject to the following constraints.[8]

Constraints on start and finish times. Constraint (2) indicates that all $S_{i,j,t}$ and $F_{i,j,t}$ are binary variables. Constraints (3) and (4) establish the relation between $s_{i,j}$ and $S_{i,j,t}$ and between $f_{i,j}$ and $F_{i,j,t}$, respectively. Note that $F_{i,j,t}$ is set to one after the activity completes; thus, we do not add one in Constraint (7). Constraint (5) sets all $S_{i,j,t}$ and $F_{i,j,t}$ to zero before the start time of the first activity of the project (if $s_{i,1}^{nom}$ is not given, we assume that the project can start at $t=1$, i.e., $s_{i,1}^{nom}=1$) [Bonfietti et al. 2014; Kolisch and Sprecher 1996; Kramer and Hwang 1991]. Constraint (6) (resp. Constraint (7)) prevents each start $S_{i,j,t}$ (resp. finish $F_{i,j,t}$) indicator variable from having a value of zero once an activity has started (resp. finished) execution [Bonfietti et al. 2014; Kolisch and Sprecher 1996; Kramer and Hwang 1991]. Constraint (8) ensures that an activity's start time precedes its finish time [Ayala and Artigues 2010; Bonfietti et al. 2014; Christofides et al. 1987; Talbot 1982].

$$\forall i, \forall j, \forall t, \quad S_{i,j,t}, F_{i,j,t} \in \{0,1\} \tag{2}$$

$$\forall i, \forall j, \quad s_{i,j} = T - \left(\sum_{t=1}^{T} S_{i,j,t}\right) + 1 \tag{3}$$

$$\forall i, \forall j, \quad f_{i,j} = T - \left(\sum_{t=1}^{T} F_{i,j,t}\right) \tag{4}$$

$$\forall i, \forall j, \forall t < s_{i,1}^{nom}, \quad S_{i,j,t} = 0, F_{i,j,t} = 0 \tag{5}$$

[8]In our description, we point to example references that adopt similar formulations.

$$\forall i, \forall j, \forall t, \quad S_{i,j,t} \geq S_{i,j,t-1} \tag{6}$$

$$\forall i, \forall j, \forall t, \quad F_{i,j,t} \geq F_{i,j,t-1} \tag{7}$$

$$\forall i, \forall j, \quad \sum_{t=1}^{T} S_{i,j,t} \geq \sum_{t=1}^{T} F_{i,j,t} \tag{8}$$

Constraint on activity precedence. Constraint (9) ensures precedence requirements: all predecessors of an activity $a_{i,j}$ must complete before its start time $s_{i,j}$ [Ayala and Artigues 2010; Baptiste and Demassey 2004; Keller and Bayraksan 2009; Kolisch and Sprecher 1996].

$$\forall i, s_{i,j} > f_{i,h}, \forall h \in H^a(i,j) \tag{9}$$

Constraint: Upper bounds on resource consumptions. Constraints (10) and (11) upper-bound the total number of resources of each type that are used at time t, summed over all activities of all projects (each project). Recall that we use $y_{i,j,k,t}$ to denote the number of conditionally-shared resources of type k that are used by activity $a_{i,j}$ of project P_i at time t. We use $z_{i,j,k,t}$ to denote the number of conditionally-shared resources of type k that are used by activity $a_{i,j}$ of project P_i at time t from the pool of unused conditionally-shared resources of other projects. That is, $z_{i,j,k,t}$ denotes the number of resources borrowed from other projects. Constraints (12) to (19) ensure that an activity does not use any resources before it starts or after it ends [Bienstock and Zuckerberg 2009; Christofides et al. 1987; Talbot 1982]. For example, Constraint (12) ensures that no resources are used before the activity starts ($S_{i,j,t} = 0, \forall t < s_{i,j}$, which forces $r_{i,j,k,t} = 0, \forall t < s_{i,j}$) and Constraint (13) ensures that no resources are used after the activity finishes ($F_{i,j,t} = 1, \forall t > f_{i,j}$, which forces $r_{i,j,k,t} = 0, \forall t > f_{i,j}$). Constraint (14) also sets an upper bound on the number of segregated resources of type k used by $a(i,j)$. Constraint (20) sets an upper bound on the total number of fully-shared resources of type k used by all activities of all projects. Constraint (21) sets an upper bound on the total number of conditionally-shared resources of type k used by all activities of P_i. Constraint (22) ensures that the total number of resources used by all the projects from the unused conditionally-shared resource pool is not greater than the number of resources available in the pool. Constraints (23) and (22) together ensure that a project does not receive resources from its own contribution to the unused conditionally-shared resource pool. The range of p is $1, \ldots, N$, and $p \neq i$. We do not include $M_{p,k,t}$ in order to use conditionally-shared resources only from other projects.

$$\forall k, \forall t, \quad \sum_{i=1}^{N} \sum_{j=1}^{J(i)} (r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq R_{k,t} \tag{10}$$

$$\forall i, \forall k, \forall t, \quad \sum_{j=1}^{J(i)} (r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq G_{i,k,t} \tag{11}$$

$$\forall i, \forall k, \forall j, \forall t, \quad r_{i,j,k,t} \leq U_{k,t} \times S_{i,j,t} \tag{12}$$

$$\forall i, \forall k, \forall j, \forall t, \quad r_{i,j,k,t} \leq U_{k,t} \times (1 - F_{i,j,t}) \tag{13}$$

$$\forall i, \forall k, \forall j, \forall t, \quad q_{i,j,k,t} \leq V_{i,k,t} \times S_{i,j,t} \tag{14}$$

$$\forall i, \forall k, \forall j, \forall t, \quad q_{i,j,k,t} \leq V_{i,k,t} \times (1 - F_{i,j,t}) \tag{15}$$

$$\forall i, \forall k, \forall j, \forall t, \quad y_{i,j,k,t} \leq M_{i,k,t} \times (S_{i,j,t}) \tag{16}$$

$$\forall i, \forall k, \forall j, \forall t, \quad y_{i,j,k,t} \leq M_{i,k,t} \times (1 - F_{i,j,t}) \tag{17}$$

$$\forall i, , \forall j, \forall k, \forall t, \quad z_{i,j,k,t} \leq \sum_{p=1}^{N} M_{p,k,t} \times S_{i,j,t} \tag{18}$$

$$\forall i, , \forall j, \forall k, \forall t, \quad z_{i,j,k,t} \leq \sum_{p=1}^{N} M_{p,k,t} \times (1 - F_{i,j,t}) \tag{19}$$

$$\forall k, \forall t, \quad \sum_{i=1}^{N} \sum_{j=1}^{J(i)} r_{i,j,k,t} \leq U_{k,t} \tag{20}$$

$$\forall i, \forall k, \forall t, \quad \sum_{j=1}^{J(i)} y_{i,j,k,t} \leq M_{i,k,t} \tag{21}$$

$$\forall k, \forall t, \quad \sum_{i=1}^{N} \sum_{j=1}^{J(i)} z_{i,j,k,t} \leq \sum_{i=1}^{N} \left(M_{i,k,t} - \sum_{j=1}^{J(i)} y_{i,j,k,t} \right) \tag{22}$$

$$\forall i, \forall k, \forall t, \quad \sum_{j=1}^{J(i)} z_{i,j,k,t} \leq \sum_{p \neq i} \left(M_{p,k,t} - \sum_{j=1}^{J(p)} y_{p,j,k,t} \right) \tag{23}$$

Constraint: Resource requirements of activities. Constraint (24) ensures the completion of an activity [Salewski et al. 1997]. One way to model heterogeneity in resource requirements is to add finer-grained activities, such as "big-block early-design-phase STA run," "medium-block late-design-phase STA run," and so on. In this example, {big, medium, small} block size × {early, middle, late} stage of design indicates growth of exceptions and corners as one transitions from early to late. We can model heterogeneous activities with different resource requirements for these activities by appropriately varying $L_{i,j,k}$.

$$\forall i, \forall k, \forall j, \sum_{t=1}^{T} (r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) = L_{i,j,k} \tag{24}$$

Constraint: Resource co-constraints. Constraint (25) ensures that the number of resources of type k used by activity $a_{i,j}$ satisfies the upper-bound constraints implied by the co-constraints between its predecessor resources (see Table II). For instance, let the number of used resources of type $k = 1$ (e.g., compute nodes) be upper-bounded by $2\times$ the number of used resources of type $k = 2$ (e.g., static timing analysis (STA) licenses) at all times for $a_{1,1}$, i.e., at most two compute nodes can be used for every STA license. Therefore, $H^r(1, 1, 1) = \{2\}$ and $g(1, 1, 2, 1, t) = 2$ at all times. The constraint will set $(r_{1,1,1,t} + q_{1,1,1,t} + y_{1,1,1,t} + z_{1,1,1,t}) \leq 2 \times (r_{1,1,2,t} + q_{1,1,2,t} + y_{1,1,2,t} + z_{1,1,2,t})$, $\forall t$. Note that this constraint is specific to each activity of a project and not for the entire project. To the best of our knowledge, previous works do not handle such co-constraints.

$$\forall i, \forall j, \forall k, \forall t, \forall h \in H^r(i, j, k),$$
$$(r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq g(i, j, h, k, t) \times (r_{i,j,h,t} + q_{i,j,h,t} + y_{i,j,h,t} + z_{i,j,h,t}) \tag{25}$$

Constraint: Stability in resource allocation. Constraints (26) and (27) ensure stability in the consumption of resources for each project. That is, we upper-bound the change in the quantity of each resource used by any given project between successive timesteps t and $t-1$. In the real world, resources such as engineers may work on activities related to multiple projects in a day. However, major changes to allocations do not,

as a practical matter, occur within short time windows. For example, if 100 engineers work on an activity of Project A for a week, reassigning 80 of them to work only on Project B in the following week would be undesirable in management's perspective.

$$\forall i, \forall k, \forall t,$$

$$\sum_{j=1}^{J(i)}(r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t})-\sum_{j=1}^{J(i)}(r_{i,j,k,t-1}+q_{i,j,k,t-1}+y_{i,j,k,t-1}+z_{i,j,k,t-1}) \leq B_{i,k,t} \quad (26)$$

$$\sum_{j=1}^{J(i)}(r_{i,j,k,t-1}+q_{i,j,k,t-1}+y_{i,j,k,t-1}+z_{i,j,k,t-1})-\sum_{j=1}^{J(i)}(r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t}) \leq B_{i,k,t} \quad (27)$$

Constraint: Tethering forecast resource allocations. (See Section 4.2.) Constraints (28) and (29) ensure that a project's forecast resource allocation is not modified by more than a certain degree (indicated by δ). Specifically, no forecast value in the active period ($s_{i,j}^{nom} \leq t \leq f_{i,j}^{nom}$) of the activity[9] can be perturbed by more than δ% in the MILP solution. Constraint (30) ensures that activity $a_{i,j}$ consumes exactly the amount of resources needed, according to the forecast resource allocation, for its completion.

$$\forall i, \forall j, \forall k, \forall\ s_{i,j}^{nom} \leq t \leq f_{i,j}^{nom}, (r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \geq w_{i,j,k,t}(1 - \delta/100) \quad (28)$$

$$\forall i, \forall j, \forall k, \forall\ s_{i,j}^{nom} \leq t \leq f_{i,j}^{nom}, (r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq w_{i,j,k,t}(1 + \delta/100) \quad (29)$$

$$\forall i, \forall j, \forall k, \sum_{t=1}^{T}(r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) = \sum_{t=1}^{T} w_{i,j,k,t} \quad (30)$$

Intuition behind the variables included in the model. We choose input parameters and optimization variables based on typical usages in IC design companies. We use $H^a(i, j)$ to enforce precedence relations among the activities of a project (e.g., parasitic extraction cannot start until the design has completed routing; STA cannot start until the design has been synthesized; or STA with signal integrity cannot start until the design has been placed). We use $R_{k,t}$ because companies typically budget for a certain number of resources during the planning phase. However, they may increase the number of resources of a particular type during the project's execution when they realize that its deadline cannot be met without these additional resources. The time-dependent variable allows us to handle such changes in our formulation. We introduce $H^r(i, j, k)$ and $g(i, j, h, k, t)$ to handle co-constraints between resource types. For instance, at most two compute nodes can be used for each STA license used. Similar to $R_{k,t}$, we use $U_{k,t}$ as the upper bound on the number of fully-shared resources, which can change over time. For example, when a project's deadline becomes risky to meet, units of resources may be removed from the shared pool and allocated to the dedicated pool of the project four work-weeks before tapeout (TO). We use $B_{i,k,t}$ to achieve a stable allocation, since resources should not be drastically shuffled ("whipsawed") across projects in consecutive units of time. For instance, we may not want to allocate 100 engineers to a project on Day 1, but only five engineers on Day 2.

Penalty functions in the objective. The objective function can be any function that is linear in the optimization variables presented in Table II. We use an objective function that minimizes the sum of two schedule-related penalties over all projects [Mohring

[9]If a schedule cannot be pulled in, then the lower bound on t should be 1 (instead of $s_{i,j}^{nom}$) in Constraints (28) and (29).

et al. 2001]. The first penalty is for the overall duration of each project relative to the nominal duration of the project. The second penalty is for the duration of each activity in each project relative to the nominal duration of the activity. Commonly used penalty functions are: **Ramp** — penalty due to each successive day of schedule slip increases linearly as we move further past the deadline (thus, the total penalty is quadratic in number of days in the slip); **Step** — penalty due to each successive day of slip is constant (and the total penalty is linear in the magnitude of schedule slip); **Delta** — total penalty for slip is constant (does not depend on the extent of the slip). We use nominal duration of the activities (and projects) to penalize the schedule. The nominal finish time of a project is calculated using the nominal start time of the first activity of the project $s_{i,1}^{nom}$ and the nominal duration of the project e_i^{nom}. The nominal finish time of $a_{i,j}$ can be calculated using the nominal start time of the activity and the nominal duration of the activity, i.e., $f_{i,j}^{nom} = s_{i,j}^{nom} + d_{i,j}^{nom}$, where $s_{i,j}^{nom} = max\{1 + f_{i,h}^{nom}\}$, over all $h \in H^a(i, j)$, or $s_{i,j}^{nom} = s_{i,1}$ if $H^a(i, j) = \emptyset$.

Complexity of the MILP. Even in a large SOC product company, number of projects $N \leq 30$, number of activities per project $J(i) \leq 20$, number of resource types $K \leq 10$, and $T \leq 300$ when the unit of time is days. There are $(2 \times N \times J(i) + 2 \times N \times J(i) \times T + 4 \times N \times K \times J(i) \times T)$ variables $(= 2 \times 30 \times 20 + 2 \times 30 \times 20 \times 365 + 4 \times 30 \times 10 \times 20 \times 365 \approx 9M$, for 365 days). We note that actual values of N, K, T, and so on will likely be smaller than these bounds. If necessary, to reduce the number of variables, we can change the unit of time from days to weeks or months. In our experiments, we use *IBM ILOG CPLEX v12.6* [CPLEX] as our solver and the runtime of our MILP is around 45s for a total of $\sim 10K$ variables, and 9min for a total of $\sim 100K$ variables, and 52min for a total of $\sim 500K$ variables (see also Figure 8).

Notice that there are two types of input scenarios that can lead to infeasible solutions.

—If the value of T (maximum duration over all projects) is not large enough for all projects to finish within that duration, CPLEX will report that the MILP is infeasible.
—Infeasibility can also arise due to inconsistent resource constraints. For example, if 20 units of resource A and 10 units of resource B are required for the completion of an activity of a project but the co-constraint is such that to use one unit of A, one unit of B must be used, infeasibility arises because we will never be able to use more than 10 units of A.

Example of SCM. We now describe the SCM problem formulation, with the help of a small example. Table III shows the values of input variables and their meaning.

Optimal solution. We seek to minimize the schedule makespan of both projects for this example. Table IV shows one of the possible optimal solutions for the example problem. Both of the projects can be completed by $t = 4$. (Resource utilization for each activity is shown only for the first resource. The utilization for the second resource is identical.) We note that $a_{1,2}$ utilizes five units of the first resource at $t = 4$ from the unused conditionally-shared pool of P_2. The formulation is able to capture the notion that if a project is not using any of its conditionally-shared resources, then those resources can be used by other active projects.

3.3. MILP Description of the Resource Cost Minimization Formulation

Given the inputs listed in Table II for the RCM formulation, we seek to minimize the total number of resources required and the total cost (i.e., sum of schedule penalties) of all projects:

$$\text{minimize} \quad \sum_{i=1}^{K} C_k \tilde{U}_k + \sum_{i=1}^{N} \sum_{t=1}^{T} C_i^p(t) + \sum_{i=1}^{N} \sum_{j=1}^{J(i)} \sum_{t=1}^{T} C_{i,j}^a(t) \tag{31}$$

Table III. Input Variables and Their Meanings for the SCM Example

Variable and value	Meaning
$N = 2$	There are two projects, P_1 and P_2.
$T = 10$	The maximum duration over both projects is 10.
$J(1) = 2$	There are two activities, $a_{1,1}$ and $a_{1,2}$, in project P_1.
$J(2) = 1$	There is one activity, $a_{2,1}$, in project P_2.
$H^a(1, 2) = 1$,	In P_1, activity a_1 must complete before activity a_2.
$K = 2$	There are two types of resources.
$R_{1,t} = 40$, $R_{2,t} = 40$	At most 40 units of either resource can be used at any point in time.
$H^r(1, 1, 2) = 1$, $H^r(1, 2, 2) = 1$, $H^r(2, 1, 2) = 1$	The first resource is a predecessor resource for the second resource for all activities and projects.
$g(1, 1, 1, 2, t) = 1\ \forall t$, $g(1, 2, 1, 2, t) = 1\ \forall t$, $g(2, 1, 1, 2, t) = 1\ \forall t$	One unit of the first resource must be used before using one unit of the second resource by any activity at any point in time.
$L_{1,1,1} = 60$, $L_{1,2,1} = 65$, $L_{1,1,2} = 60$, $L_{1,2,2} = 65$, $L_{2,1,1} = 30$, $L_{2,1,2} = 30$	60 units of each resource are required to complete activity $a_{1,1}$, 65 units of each resource are required to complete activity $a_{1,2}$, and 30 units of each resource are required to complete activity $a_{2,1}$.
$U_{1,t} = 20\ \forall t$, $U_{2,t} = 20\ \forall t$	At most 20 units of each resource are fully-shared between both projects at any point in time.
$V_{1,1,t} = 5$, $V_{1,2,t} = 5\ \forall t$, $V_{2,1,t} = 5$, $V_{1,2,t} = 5\ \forall t$	Each project has five units of each resource that are segregated, i.e., they can be used only by activities of that project at any point in time. These resources are not shared with other projects.
$M_{1,1,t} = 5$, $M_{1,2,t} = 5\ \forall t$, $M_{2,1,t} = 5$, $M_{1,2,t} = 5\ \forall t$	Each project has five units of each resource that are conditionally-shared at any point in time, i.e., they can be used by activities of other projects if they are unused by the project.
$G_{1,1,t} = 35$, $G_{1,2,t} = 35\ \forall t$, $G_{2,1,t} = 35$, $G_{1,2,t} = 35\ \forall t$	At most 35 units of either resource can be used by either project at any point in time.

Table IV. Consumption of the First Resource for Both Projects in an Optimal Solution

	Activities											
	$a_{1,1}$				$a_{1,2}$				$a_{2,1}$			
Time t	$r_{1,1,1,t}$	$q_{1,1,1,t}$	$y_{1,1,1,t}$	$z_{1,1,1,t}$	$r_{1,2,1,t}$	$q_{1,2,1,t}$	$y_{1,2,1,t}$	$z_{1,2,1,t}$	$r_{2,1,1,t}$	$q_{2,1,1,t}$	$y_{2,1,1,t}$	$z_{2,1,1,t}$
1	20	5	5	0	0	0	0	0	0	5	5	0
2	20	5	5	0	0	0	0	0	0	5	5	0
3	0	0	0	0	20	5	5	0	0	5	5	0
4	0	0	0	0	20	5	5	5	0	0	0	0
Total (for each activity)	60				65				30			

This optimization is subject to the following constraints.

Constraints on start and finish times. We use Constraints (2) to (8), as in the SCM formulation (Section 3.2).

Constraint on activity precedence. We use Constraint (9), as in the SCM formulation (Section 3.2).

Constraint: Upper bounds on resource consumptions. Constraints (32) and (33) upper-bound the number of resources of each type used at time t across all activities of all projects. Constraints (34) and (35) ensure that an activity does not use any resources

before it starts or after it ends.

$$\forall i, \forall k, \forall t, \sum_{j=1}^{J(i)} r_{i,j,k,t} \leq G_{i,k,t} \tag{32}$$

$$\forall k, \forall t, \sum_{i=1}^{N} \sum_{j=1}^{J(i)} r_{i,j,k,t} \leq \tilde{U}_k \tag{33}$$

$$\forall i, \forall k, \forall j, \forall t, \quad r_{i,j,k,t} \leq G_{i,k,t} \times S_{i,j,t} \tag{34}$$

$$\forall i, \forall k, \forall j, \forall t, \quad r_{i,j,k,t} \leq G_{i,k,t} \times (1 - F_{i,j,t}) \tag{35}$$

Constraint: Resource requirements of activities. Constraint (36) ensures the completion of an activity.

$$\forall i, \forall k, \forall j, \sum_{t=1}^{T} r_{i,j,k,t} = L_{i,j,k} \tag{36}$$

Constraint: Stability in resource allocation. We modify Constraints (26) and (27) from the SCM formulation as follows to ensure stability in the consumption of resources for each project.

$$\forall i, \forall k, \forall t, \sum_{j=1}^{J(i)} (r_{i,j,k,t}) - \sum_{j=1}^{J(i)} (r_{i,j,k,t-1}) \leq B_{i,k,t} \tag{37}$$

$$\forall i, \forall k, \forall t, \sum_{j=1}^{J(i)} (r_{i,j,k,t-1}) - \sum_{j=1}^{J(i)} (r_{i,j,k,t}) \leq B_{i,k,t} \tag{38}$$

4. VALIDATION AND RESULTS

In this section, we describe computational studies using three multi-project scheduling problem instances taken from a large design center (tens of market-leading SOC product tapeouts per year) of a worldwide top-5 semiconductor company, referred to from here as Company X. The results show a potential for significant resource savings (datacenter provisioning, EDA tool licenses, people, and schedule) from our MILP formulations when compared to the scheduling solutions actually used by Company X's design center. We also show the scaling of solver runtime with instance parameters.

4.1. Schedule Modification Use Case

The first industry problem instance has $N = 3$ projects, each in the final pass of implementation, within an overall timeline of $T = 90$ days. The three projects P_1, P_2, and P_3 contain 15, 10, and 10 "hard macro" blocks, respectively. As listed in Table V, there are 11 activities associated with each project ($a_{i,1} = \text{A1}, \ldots, a_{i,11} = \text{A11}$).[10] The table shows that each activity, *per block*, uses some amount of each of five resource types: compute cores, units of memory (e.g., a unit might be 16GB RAM), and tool licenses of types L1, L2, and L3.[11] Further, activities A5, A8, and A11 per block are

[10]See Table IX in the Appendix for a mapping of activities and resources to actual chip design flow terminologies.

[11]According to Gary Smith personal communication [], leading exemplars of these resources include EDA tools such as Cadence's *Innovus* [Innovus], *Assura QRC* [Assura QRC] and *Tempus Timing Signoff* [Tempus], and Synopsys's *IC Compiler* [IC Compiler], *Star-RCXT* [StarRCXT] and *PrimeTime-SI* [PrimeTime]. The

Table V. Activity Requirements (Per Block) for Each Project

Activity	#core	#mem	L1	L2	L3	Hours
1. A1	1	1	1			12
2. A2	4	2	2			24
3. A3	4	2	2			72
4. A4	4	2		1		8
5. A5 (per corner)	8	8			1	4
6. A6	4	4			1	12
7. A7	4	2		1		8
8. A8 (per corner)	8	8			1	4
9. A9	8	8		1	1	24
10. A10	4	2		1		8
11. A11 (per corner)	8	8			1	4

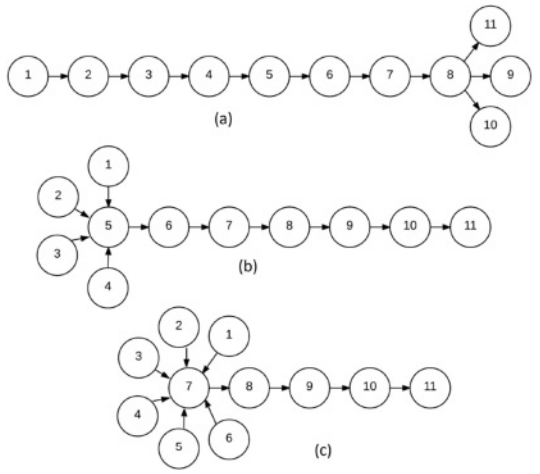


Fig. 2. Precedence order of activities in projects (a) P_1, (b) P_2, and (c) P_3.

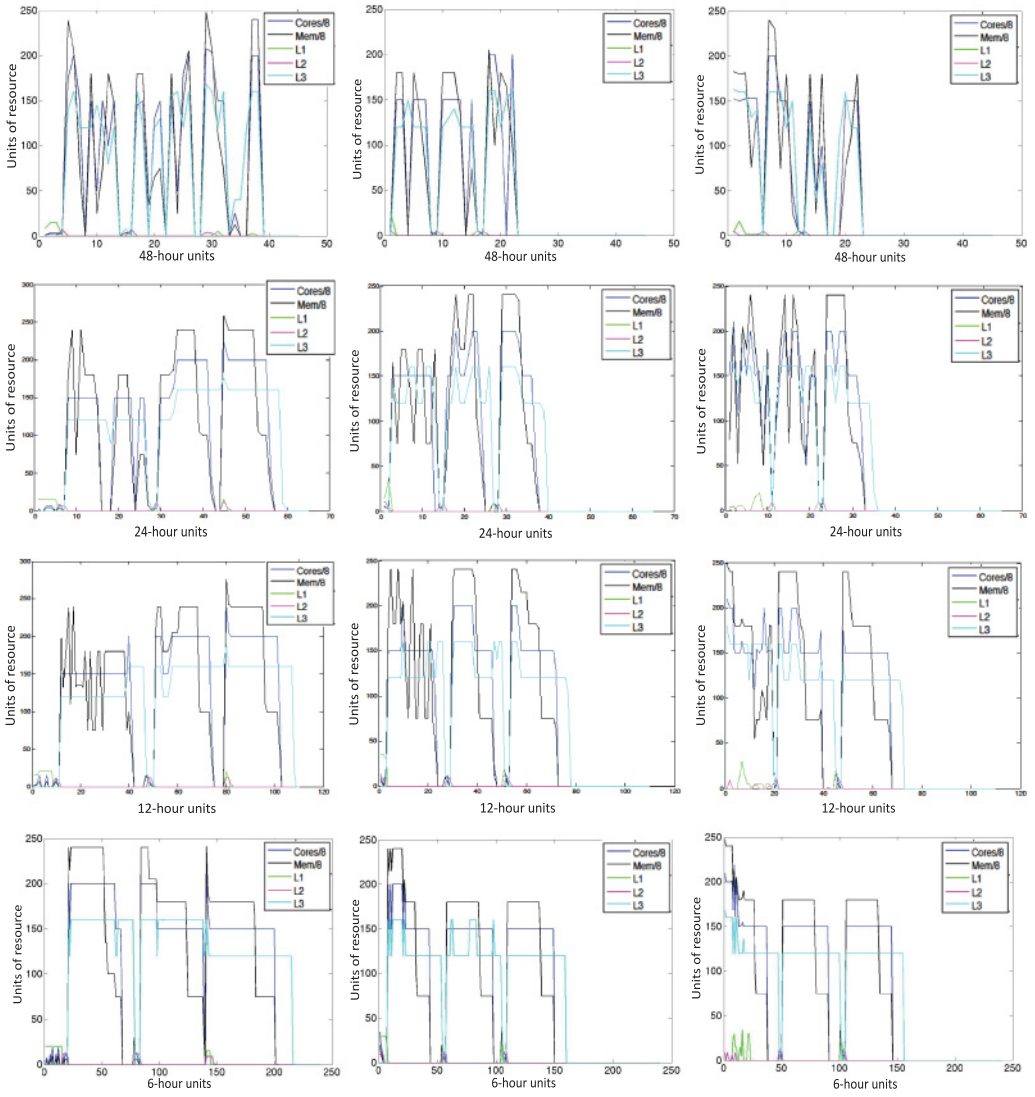
performed at 75 corners and two modes (functional and test).[12] The projects have access to the following total amounts of these resources: (i) compute cores = 4800, (ii) units of memory = 4800, (iii) L1 licenses = 50, (iv) L2 licenses = 30, and (v) L3 licenses = 400.[13]

Additional constraints governing the projects and the scheduling solution are as follows: (i) for each project, the activities must follow a given precedence order, as shown in Figure 2 – for example, in Project P_2, activities $a_{2,1}$, $a_{2,2}$, $a_{2,3}$, and $a_{2,4}$ must all be completed before activity $a_{2,5}$ can commence, but there are no ordering constraints among $a_{2,1} - a_{2,4}$; (ii) at any point in time, the number of compute cores consumed cannot exceed 10 times the number of tool licenses consumed and cannot exceed twice the number of units of memory consumed; (iii) each project is given a 30% allocation of the 4800 total compute cores (i.e., as segregated resources), with the remaining 10% of the compute cores being fully-shared resources; and (iv) no project can use more than

EDA tools used in the production design flows studied in this article cannot be specifically revealed here but are from this set.

[12]Thus, for example, performing A5, A8, or A11 activity for a 10-block chip will require 10 (blocks) × 75 (corners) × 2 (modes) × 8 (cores) × 4 (hours) = 48000 core-hours of compute resource.

[13]In this instance, there are ∼65K variables and ∼980K constraints. The runtime of the solver is around 9min.



(a) Project P_1 (b) Project P_2 (c) Project P_3

Fig. 3. MILP solutions for projects (a) P_1, (b) P_2, and (c) P_3 at 48h, 24h, 12h, and 6h granularities from top to bottom, respectively. For readability, we have scaled down the values of cores and storage memory as Cores/8 and Mem/8.

350 L3 licenses, 40 L1 licenses, or 60% of the total supply of any other type of resource (compute cores, memory units, L2 licenses) at any time.

MILP solution using SCM. Our SCM MILP formulation straightforwardly allows capture of the multi-tapeout project scheduling problem described earlier. All projects can be completed within the 90-day limit. In one optimal solution, projects P_1, P_2, and P_3 are completed in 59, 39, and 34 days, respectively. Figures 3(a) to 3(c) show the resource consumption profiles of the three projects, where no stability constraints, i.e., Constraints (26) and (27), are imposed. From top to bottom, the schedules for each project are shown at 48h, 24h, 12h, and 6h granularities, respectively. When

timestep granularity is coarsest (48h), MILP runtime is around 3.4min and resource consumptions switch rapidly across activities, whereas when timestep granularity is finest (6h), MILP runtime is around 2.2h and the makespan tightens as compared to the makespan of 48h granularity solutions.

Schedule modification. The salient problem that we address in this section is one of late-breaking schedule changes that can introduce iterations of activities in a project (e.g., synthesis, verification, placement, routing and sign-off). After projects are initially scheduled, there can arise a need to modify some of the instance parameters during schedule execution (e.g., due to a design bug and resulting Engineering Change Order (ECO)), then re-solve for the project schedules from that point on. Here, a late-breaking bug (i.e., a bug that is found and fixed very late in the schedule) in the behavioral description of the design for project P_2 caused large-scale changes. In the actual project, this led to a push-out of activity $a_{2,8}$ (A8), which, in turn, pushed out all downstream activities for the project. As a result, there is a need to determine optimal scheduling of the remaining activities, i.e., where P_1 resumes from $a_{1,5}$ on, P_2 resumes from $a_{2,8}$ on, and P_3 has only its last activity $a_{3,11}$ remaining to be scheduled. An optimal MILP solution for the "from the moment of the ECO onward" scheduling problem is shown in Figures 4(a), (c), and (e) with projects P_1, P_2, and P_3. The solution actually used in the company design center is shown in Figures 4(b), 4(d), and 4(f). In the MILP solution, all three projects are completed by 34 extra days from the point of the late-breaking bug, while the industry solution takes 41 extra days for completion. Our MILP solution could thus have saved 1.4 work-weeks in the schedule makespan of the three projects.[14]

4.2. Scheduling Tethered to Forecasts

The SCM MILP formulation can be extended, with a few additional constraints (and corresponding inputs), to address a *forecast-tethered* resource allocation problem. The use case is that we are given (typically, bottoms-up from project owners) a forecast resource allocation for activity $a_{i,j,k}$ and its consumption $w_{i,j,k,t}$ of resource type k. The optimal solution must satisfy the Constraints (28) to (30). Figure 5(a) illustrates the allocation for one project; Figure 5(b) illustrates a consumption forecast over time for three identical such projects. At times, forecast consumption is greater than the upper bounds of resources (e.g., servers/datacenter capacity); thus, the allocation is infeasible. Figure 5(c) shows a feasible scheduling that is obtained by modifying the forecast resource allocation within upper bounds, constraining the consumption peaks to be within bounds.

We find an optimal schedule by tethering an instance of an industrial forecast resource allocation from the design center of Company X. The instance consists of 24 projects along with the forecast resource consumption of each project from November 2014 to September 2015. The total forecast resource consumption over all the projects is greater than the current servers (and datacenter capacity) during certain months. Therefore, we optimize the allocation in order to bound the consumption within $R_{k,t}$ (i.e., the current servers or datacenter capacity). We consider two variants: (i) pull-in of the project schedule and (ii) reduction of the amounts of shared allocations from the upper bound $R_{k,t}$. Table VI summarizes the experiments that we conduct for this instance. $CS = 1560$ denotes the number of current servers and $DC = 2100$ denotes the datacenter capacity. *fs-in* denotes whether the fully-shared resources (210 units of CS or DC)

[14]According to the actual industry solution, each of the projects P_2 and P_3 should be given 50% of the resources until P_3 is completed. This entails that P_1 will be given 50% of the resources while P_2 and P_3 (for cleanup) get 20% of the resources each, and the number of fully-shared resources is restored to 10%. Since we do not consider cleanup activity (of P_3) to get the optimized solution, we re-allocate P_3's resources to P_1 (10%, as no project can consume more than 60% of the resources) and P_2 (the remaining 10%) for fair comparison of the solutions.

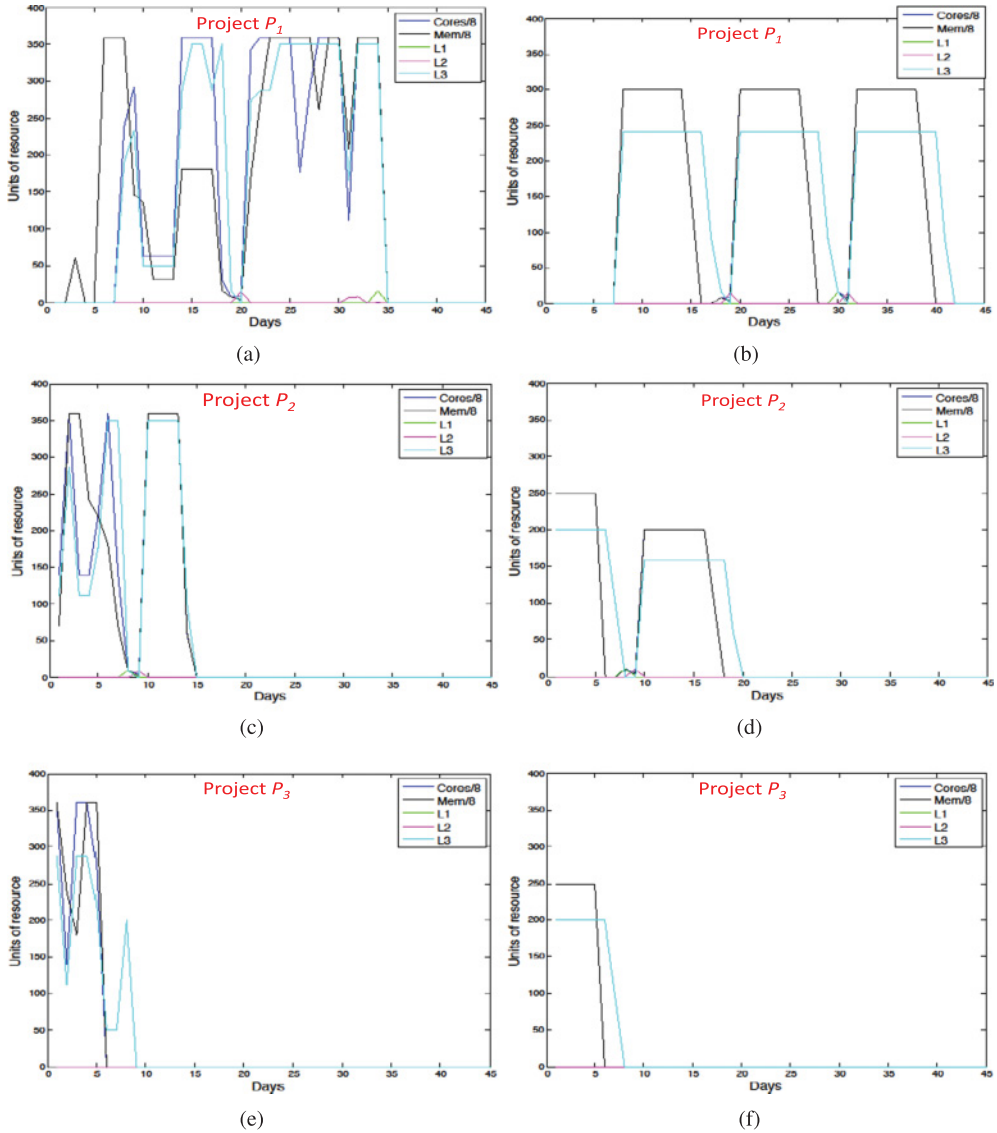


Fig. 4. Solutions of (a) MILP for P_1, (b) industry for P_1, (c) MILP for P_2, (d) industry for P_2, (e) MILP for P_3, and (f) industry for P_3 with a late-breaking RTL bug in project P_2. Cores and Mem are exactly overlapping in the industry solution in all three figures. For readability, we have scaled down the values of cores and storage memory as Cores/8 and Mem/8.

are included in $R_{k,t}$; *pi-en* denotes whether pull-in is enabled; *pi* denotes the number of months by which the schedule is pulled in; and *po* denotes the number of months by which the schedule is pushed out. Our penalty functions for schedule changes (pull-in or push-out) per-month are as follows: no penalty when the change is $<5\%$ of the forecast duration of the project; penalty function pen_1 for changes between 5% and 30% of the duration; and penalty function pen_2 for changes beyond 30% of the duration. Usually, pen_2 is significantly higher than pen_1. Furthermore, there are two types of projects – *committed* and *proposed*. Committed projects are penalized more than proposed projects

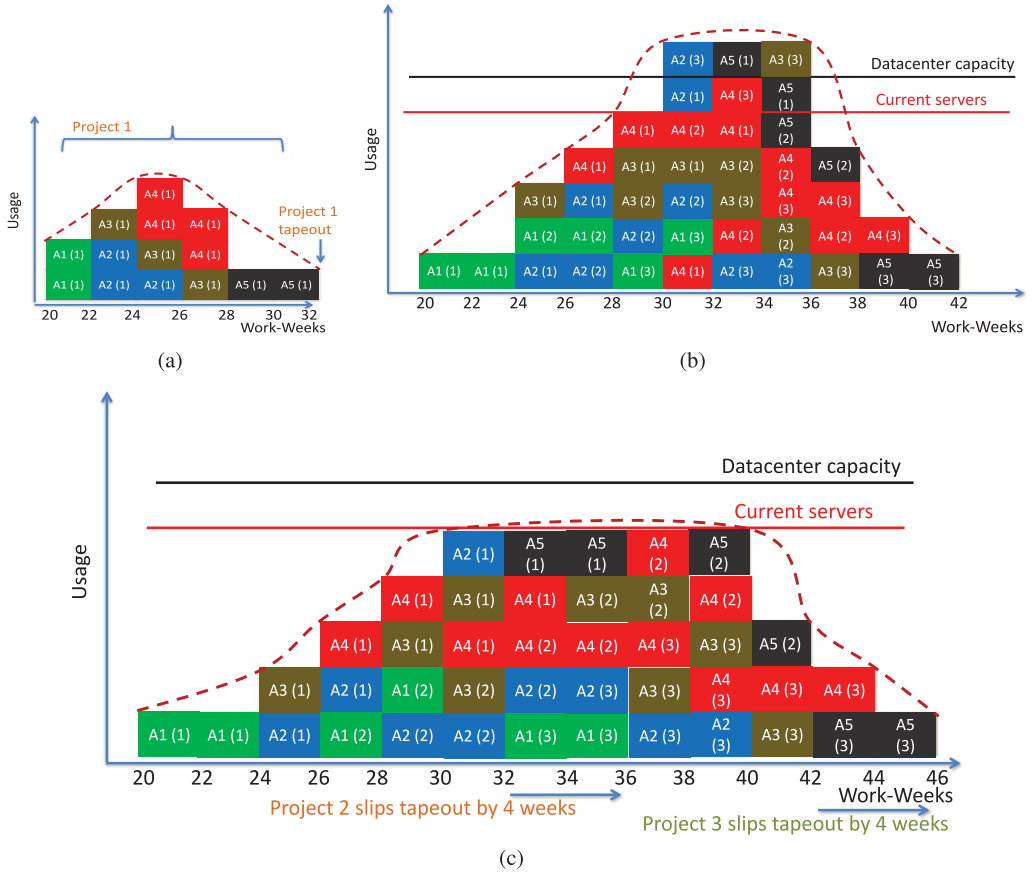


Fig. 5. Forecast allocation for (a) one project, (b) three such projects – infeasible, and (c) a feasible MILP-derived allocation.

Table VI. Resource Allocations Tethered to Forecasts

$R_{k,t}$	δ (%)	*fs-in*	*pi-en*	pen_1	pen_2	#*po*	#*pi*
CS	30	✓	✗	ramp	step	6 (3)	-
CS	30	✗	✗	infeasible			
CS	40	✗	✗	ramp	step	14 (8)	-
CS	40	✓	✗	ramp	step	3 (0)	-
CS	30	✓	✗	step	delta	15 (8)	-
DC	30	✓	✗	ramp	step	**0 (0)**	-
DC	30	✗	✗	ramp	step	**0 (0)**	-
CS	30	✓	✓	ramp	step	5 (2)	5 (0)

when not adhering to the forecast schedule.[15] Values in parentheses show the total number of months that the committed projects are either pushed out or pulled in.[16]

[15]We do not leverage our MILP currently for industrial projects. However, in our next version of our (company's) forecast reporting/methodology, we are considering integration of features described in this article to afford more "knobs to turn" when setting schedules, as there is a continuous and pressing need to achieve increased efficiencies within the compute environment.

[16]In this instance, there are ~1K variables and ~4.5K constraints. The runtime of the solver is around 3.2s.

Table VII. Billable Man-Weeks for Each Activity for Each Project

		Project			
Activity	**Resource type used**	**Project P_1**	**Project P_2**	**Project P_3**	**Project P_4**
1. A12	HR1	140	145	45	160
2. A13	HR1	420	425	45	500
3. A14	HR2	115	100	45	200
4. A15	HR2	345	300	145	580
5. A16	HR3	870	990	140	640
6. A17	HR3	80	260	30	50
7. A18	HR2	220	300	90	390
8. A19	HR4	480	550	180	540

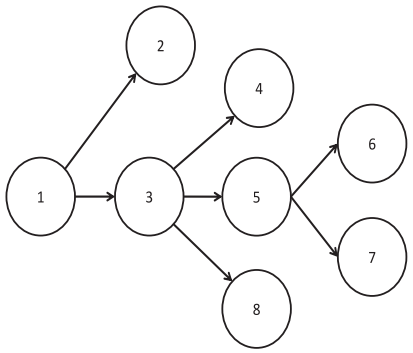


Fig. 6. Activity precedence for all projects.

Note that for the allocation to be bounded by $R_{k,t}$, δ must be sufficiently large such that tethering can bring the total consumption for each of the months to be within $R_{k,t}$. For example, if the total forecast consumption for a month is 100 units and $R_{k,t}$ is 70 units, then $\delta \geq 30\%$ to obtain a feasible solution. The maximum CPU time needed to solve any of the instances in Table VI is less than a second, since the unit of T is months and $T = 11$. The design center management of Company X could not solve this problem. Their solution was to purchase the additional 600 servers required to meet committed project forecast demands during the months of peak execution. However, we demonstrate that our solver can provide an allocation that does not require the purchase of additional servers while still meeting the schedule.

4.3. (Human) Resource Allocation Use Case

Our third industry instance has $N = 4$ projects, each with a makespan of 16 work-weeks (or 80 days). Each of the four projects P_1, P_2, P_3, and P_4 has eight activities with assigned "billable man-weeks," i.e., total amount of human resources needed to complete each activity. Four types of human resources (HR1, . . . , HR4) are available for each project. Table VII shows the resource requirement for each project across multiple activities.[17] Project P_4 begins first; the start dates of projects P_3, P_2, and P_1 are offset by 5, 9, and 5 work-weeks, respectively (there are 5d in a work-week) relative to the start date of project P_4. The precedence graph for activities for each project is shown in Figure 6. In addition, all resources are fully-shared across the four projects.[18]

[17]Table IX in the Appendix provides a mapping of activities and resources to chip design flow terminologies.
[18]In this instance, there are ∼6K variables and ∼31K constraints. The runtime of the solver is around 18s.

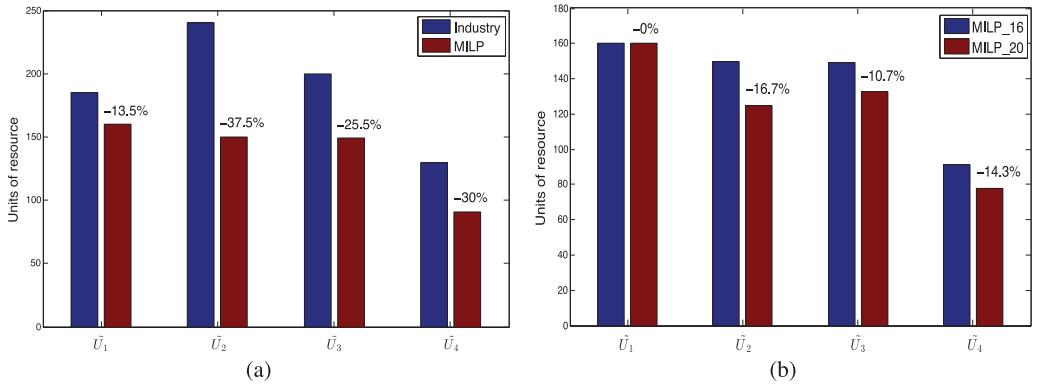


Fig. 7. Comparison of allocations: (a) Industry vs. MILP with the makespan of all projects set to 16 work-weeks, and (b) MILP solutions when the makespan of all projects is 16 vs. 20 work-weeks. $\tilde{U}_{i=1,2,3,4}$ is the maximum over total amount of HR{1,2,3,4} units consumed in any work-week.

From a scheduling standpoint, design management would typically like to assess time-to-market versus resource costs. Our solver allows "what-if" analyses to understand these trade-offs. To understand resource costs when time-to-market is critical, we set the makespan of each project to 16 work-weeks. Figure 7(a) compares our RCM MILP solution with the industry solution. The RCM MILP solution reduces the maximum amount of resources required in any work-week, $\tilde{U}_1$ (for HR1) by 13.5% (185 units to 160 units), $\tilde{U}_2$ (for HR2) by 37.5% (240 units to 150 units), $\tilde{U}_3$ (for HR3) by 25.5% (200 units to 149 units), and $\tilde{U}_2$ (for HR4) by 30% (130 units to 91 units). Such reduction in the number of human resources required can result in highly significant cost savings for a company. For example, according to Glassdoor [], one unit of an HR resource costs \$40 (US dollars) per hour in a (South Asian) non-US location. We assume this cost and that a given resource works 8 hours per day for 5 days per week. The overall makespan of all four projects is 26 work-weeks. Therefore, reducing $\tilde{U}_2$ by 90 units for HR2 saves $40 \times 8 \times 5 \times 26 \times 90 \sim \$3.7$ million for the company. To understand resource costs when time-to-market can be relaxed, we have evaluated a solution in which we set the makespan of each project to 20 work-weeks. Figure 7(b) compares MILP solutions for 20 work-weeks with those for 16 work-weeks. The relaxed project makespans enable further reductions in the maximum amount of resources required in any work-week: $\tilde{U}_3$ (for HR3) by 10.7% (149 units to 133 units), $\tilde{U}_4$ (for HR4) by 14.3% (91 units to 78 units), and $\tilde{U}_2$ (for HR2) by 16.7% (150 units to 125 units) relative to our solutions for the 16 work-week project makespan. The additional reduction of $\tilde{U}_2$ by 25 units for HR2 alone can result in further savings of $(40 \times 8 \times 5 \times 26 \times 150) - (40 \times 8 \times 5 \times 29 \times 125) \sim \$0.44$ million for the company.

4.4. Artificial Test Case Generator

We have separately developed a generator of random multi-tapeout project scheduling instances, in which parameters such as N, T, $J(i)$, $R_{k,t}$, $U_{k,t}$, $V_{i,k,t}$, $M_{i,k,t}$, $G_{i,k,t}$, and d_{nom} (see Table II) are all Gaussian random variables, and various pairs of resources may be co-constrained. Further, the randomly generated instances can have different topologies of precedence constraints (see Figure 2). Our generator is implemented in *Python*; it takes in values of N, T, $J(i)$, $R_{k,t}$, $U_{k,t}$, $V_{i,k,t}$, $M_{i,k,t}$, $G_{i,k,t}$, and d_{nom} as command-line arguments, and generates a problem instance input as illustrated in the example in Section 3.2. Large IC design companies typically deal with up to ~30 projects with known priorities (set by marketing teams and management). It is not required to study

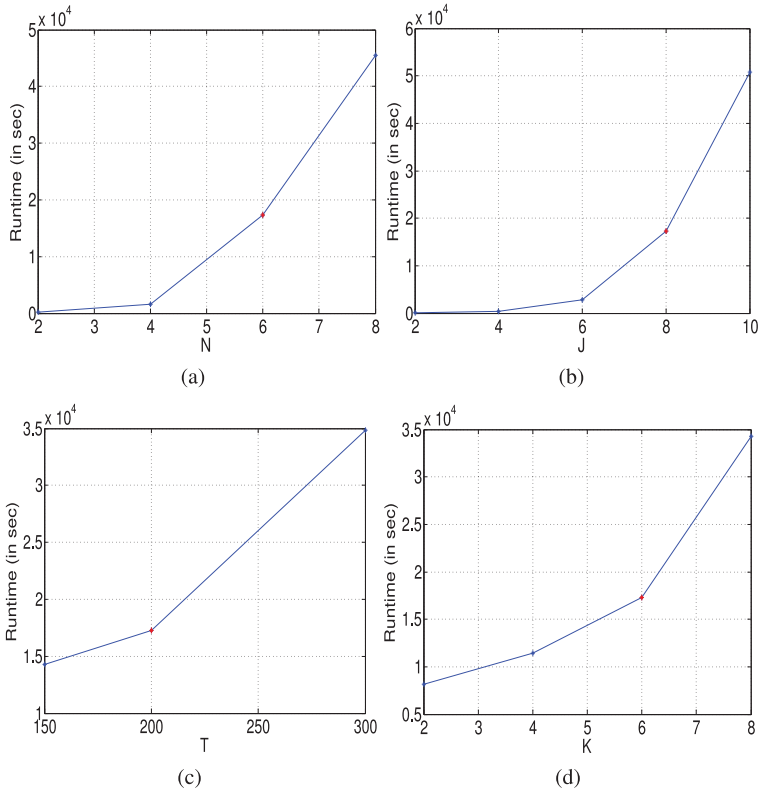


Fig. 8. Runtime variation with parameters: (a) N, (b) $J(i)$, (c) T, and (d) K.

all $N!$ permutations of projects because project priorities induce a chain ordering. As a result, unlike Kolisch and Sprecher [1996], we do not exhaustively enumerate instances. As demonstrated earlier, our MILP can handle ≤ 30 simultaneous projects whose priorities have already been decided.[19]

4.5. Scalability Studies

Furthermore, we have also studied the scalability of our optimal solution approach with respect to *CPLEX v12.6* solver runtimes. We use artificial test cases from our generator described in Section 4.4 for our scalability studies. Figure 8 shows the sensitivity of CPLEX runtime to changes in various instance parameters relative to a base instance configuration of $N = 6,\ J = 8,\ T = 200,\ K = 6$ (the red point shown in each of the plots in the figure). Each plot sweeps one of the instance parameters as: (i) $N = \{2, 4, 6, 8\}$, (ii) $J = \{2, 4, 6, 8, 10\}$ ($J(i) = J \forall i = \{1, \ldots, N\}$), (iii) $T = \{150, 200, 300\}$ days, and

[19]We have run our solver on 480 test cases for the *j30* benchmark from PSPLIB [PSPLIB], for which optimal solutions for each test case are available. Other benchmarks such as *j60*, *j90*, and *j120* do not have optimal solutions posted in PSPLIB [PSPLIB]. This benchmark has one project with 30 activities, and each test case varies the following: (i) precedence constraints between activities and (ii) upper bounds of resources for each activity over various timesteps. For each test case, renewable resources map to $R_{k,t}$, and non-renewable and doubly-constrained resources map to $G_{i,k,t}$ in our MILP. For non-renewable resources, we set the same upper bound for all t. We have confirmed that MILP solutions are the same as optimal solutions posted in PSPLIB [ ] for *j30*. The runtimes of both PSPLIB and MILP solutions differ by $\pm 3\%$ when the respective solver implementations (in *CPLEX v12.6*) are run on an Intel Xeon E5-1410 server at 2.80GHz.

(iv) $K = \{2, 4, 6, 8\}$. All other parameters are fixed at the base configuration values. The parameter being varied is shown on the x-axis; runtime (seconds) is shown on the y-axis. Here, all projects are identical, i.e., they all have the same number of activities, and corresponding activities have the same resource requirements.

4.6. Stochasticity Studies

The default usage model of the MILP solver assumes a "perfect world with no stochasticity," that is, there is some kind of optimal offline solution possible. However, in a "real world that has stochasticity," the optimal offline solution will actually result in some kind of distribution (which depends on the stochasticity that is assumed) of outcomes (e.g., schedule makespans, schedule slippage penalties, and so on). The slippages in the "real world that has stochasticity" can be mitigated by re-running the solver at intervals to adapt the scheduling solution to deviations from assumed idealities. There is an expected trade-off between makespan and penalties versus the interval between consecutive re-runnings of the solver. We conduct experiments to show that our results have the expected sensitivities to the assumed variances in the modeling of stochasticity. We modify our instance generator in the following way. Each activity can be repeated with a certain probability distribution, which is provided by the user. The resource requirements of each activity $L_{i,j,k}$ are sampled from a normal distribution $N(\mu', \sigma')$ provided by the user. $L_{i,j,k}$ with distribution $N(\mu', \sigma')$ models "productivity" of assigned resources (e.g., Engineer A is expected to complete a certain task in two days but the engineer's completion time has a standard deviation of 0.4d). When an activity is repeated, we follow the precedence graph so that dependent activities are also repeated. At various timesteps in a project's execution (i.e., defined by the interval between re-runnings of the solver), we (i) use our (stochastic) instance generator to create instances based on sampling from distributions, (ii) induce a remaining project scheduling problem instance after applying the current (previously-completed) solution, and (iii) re-run our solver on this induced instance.

We test our stochasticity model using an instance with two projects, four activities per project, linear precedence of all activities within a project, and four types of resources. We sample $L_{i,j,k}$ from $N(\mu', \sigma')$ and assume that each activity has a probability of 0.15 of being repeated. Repetition corresponds to the "anomaly" of a floor plan change, logic ECO, failed P&R run, and so on. The timestep is days in our instances, and we conduct 100 trials of each experiment and report the average makespan across these 100 trials. Figure 9(a) shows the impact on makespan as the time when anomaly happens is varied. We set the probability of each activity being repeated to 0.15, $L_{i,j,k}$ to the mean value, vary the time at which we sample each activity that has potential to be repeated at $t = \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$, and assume that the solver is re-run as soon as the anomaly (i.e., repetition of the activity) is detected. We observe that if the anomaly happens later during execution, the project makespan gets worse as compared to the anomaly happening earlier during execution (less opportunity to work around the schedule slip). We implement an oracular solver (that knows when anomaly will happen and its impact on iterations of activities at $t = 0$) and compare solutions from the oracular solver and solutions from the solver with and without cognizance of anomaly. The important point to note here is that the makespan of the oracular solution is less than or equal to the makespan of the solution when the solver is run whenever anomaly occurs.

In Figure 9(b), we study the impact on makespan when the solver's re-running interval is varied across $\xi = \{1, 2, 4, 6, 8, 10, 12, 14, 16\}$ timesteps (days) assuming that the anomaly occurs at $t = 10$. We set the probability of each activity being repeated to 0.15 and $L_{i,j,k}$ to the mean value. As expected, makespan increases when the solver is re-run much later after the anomaly happens, whereas when the solver is re-run every

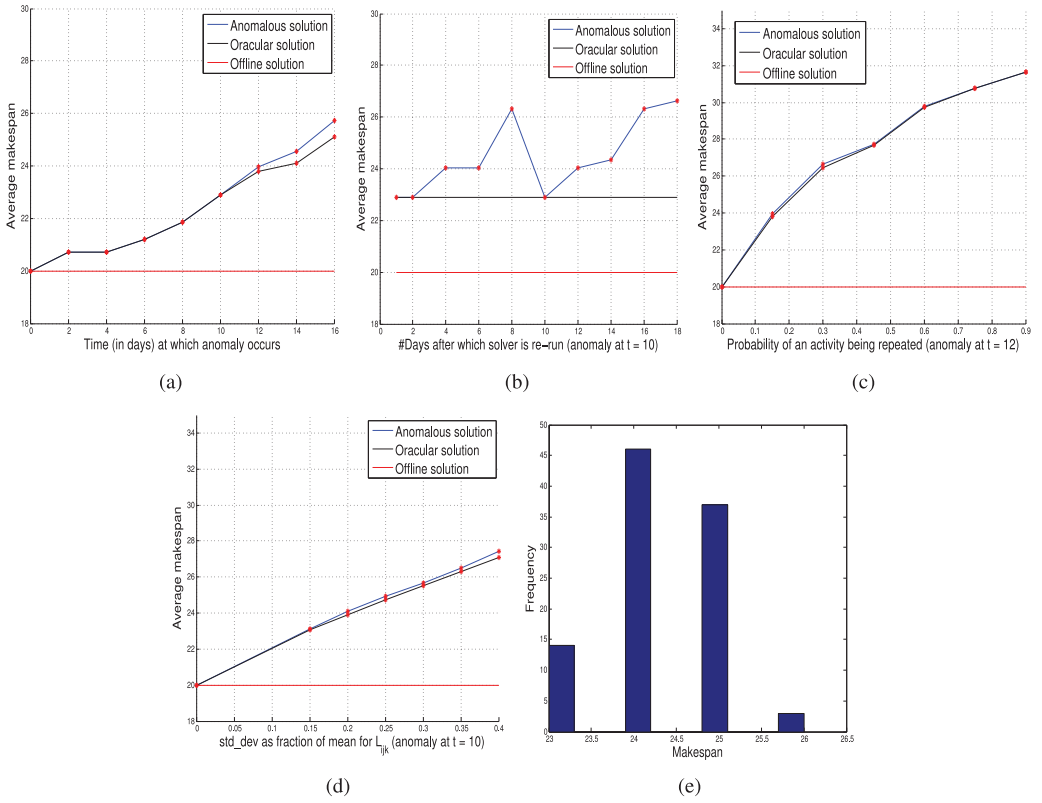
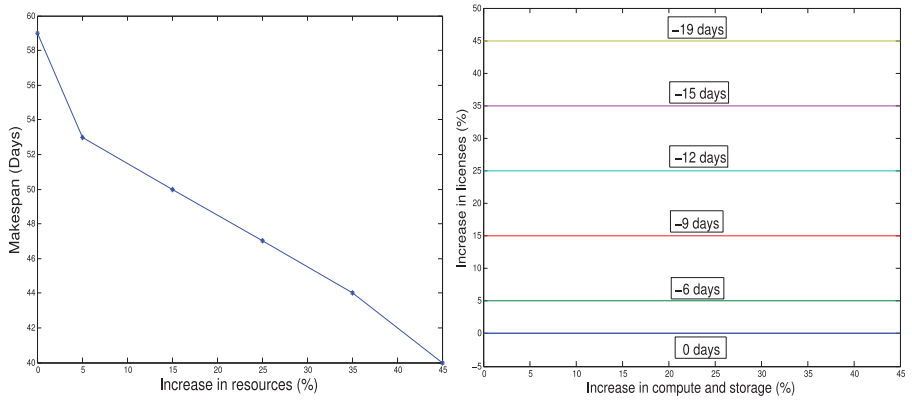


Fig. 9. Studies of stochasticity on makespan: (a) time when anomaly happens is varied, (b) interval at which the solver is re-run is varied, (c) probability of each activity being repeated is varied, (d) resource requirements are varied, and (e) when there is uncertainty in schedule.

day or every 2d or every 10d, the anomaly is detected immediately and there is no impact on makespan as compared to solutions from the oracular solver. For example, when the solver is run every 8d, anomaly is detected at $t = 16$ and the makespan diverges from the oracular solution by 3.5d. However, when the solver is run every 10d, the anomaly is detected at $t = 10$ and the makespan is the same as that from the oracular solver. In Figure 9(c), we study sensitivity of the makespan to stochasticity in repeating activities. We vary the probability of each activity being repeated from $\{0.15, \ldots, 0.9\}$ in steps of 0.05. We assume that the anomaly happens at $t = 12$ (with probability $\{0.15, \ldots, 0.9\}$), the solver is re-run every six (or any divisor of 12) days, and set $L_{i,j,k}$ to the mean value. As expected, the makespan increases as the probability of repetition increases, and solutions from the oracular solver have a smaller makespan compared to the solutions from the solver that is re-run every 6d. In Figure 9(d), we study sensitivity of the makespan to stochasticity in resource requirements when the anomaly happens (i.e., we sample resource requirements for remaining activities) at $t = 10$ and the solver is re-run immediately. We sample $L_{i,j,k}$ for activities that are running or have not yet started. We vary the standard deviation across $+\{0, 15, 20, 25, 30, 35, 40\}\%$ of the mean. We observe that as standard deviation increases (i.e., resources become less predictable) the average makespan increases. In Figure 9(e), we study distribution of the makespan when there is uncertainty in the schedule. For a given problem instance, considering that there are no anomalies, we sample $L_{i,j,k}$ from



(a) (b)

Fig. 10. (a) Impact on the makespan when the upper bounds on resources are increased. (b) Pareto curves for changes to resource upper bounds.

a distribution that has the mean set to the value of $L_{i,j,k}$ when there are no anomalies and has a standard deviation equal to +15% of the mean. We then plot the makespan for various samples of $L_{i,j,k}$. We observe that there is a minimum-makespan schedule from 100 sampled values of $L_{i,j,k}$.

Schedule Granularity. Our problem formulation discretizes time, with the unit t potentially representing hours, days, months, and so on. A more granular time unit permits more accurate modeling at the cost of runtime. Recall that Figures 3(a) to 3(c) showed solutions to the industrial schedule modification instance by varying timesteps at 48h, 24h, 12h and 6h granularities from top to bottom, respectively. As expected, we see slightly tighter makespans for the projects at the finest granularity (6h) than at the coarsest granularity (48h). The solver runtime increases from 3.4s (with 48h solution granularity) to 2.2h (with 6h solution granularity).

4.7. Sensitivity Studies

We analyze the effect of the upper bounds (on the resources) on the optimal schedule. In the first industry instance described in Section 4.1, we increase the upper bounds of the resources, that is, we proportionately increase $R_{k,t}$, $U_{k,t}$, $V_{i,k,t}$, and $G_{i,k,t}$. Figure 10(a) shows that the makespan decreases when the upper bounds of *all* the resources increase.

We also create another instance in which we increase only upper bounds of compute server and storage resources and do not change the upper bounds of other resources. From the Pareto curves in Figure 10(b), we can see that in this instance at Company X, additional compute servers and storage would not help *at all*. In this particular instance, the number of licenses is the bottleneck and the makespan can be improved only if more licenses are made available. (This also shows effects of the resource co-constraints, i.e., additional compute and storage resources cannot be maximally utilized due to shortage of licenses.) From our interactions with senior management at the Company X design center, we understand that these types of sensitivity analyses can be very useful for resource planning and procurement.

4.8. Engineer Allocation Studies

Last, we study *stable allocation* of certain resource types such as engineers, who cannot be rapidly re-allocated to different projects or activities. We modify the objective of our

Table VIII. Total Cost, Number of Switches, μ and σ of Switches Overall Engineers, and Overall Schedule Makespan Impact

Test case	N	# Engineers	Switching Cost C	Total # Switches	μ (# Switches)	σ (# Switches)	Total Cost	Δ Overall Makespan
E_1	5	100	0	4108	39.8	7.33	0	0 weeks
			0.1	3804	38.8	7.20	410.4	+1 weeks
			1	2180	21.5	6.36	1920	+1 weeks
			10	1280	12.4	4.28	12840	+1 weeks
			100	0	0	0	15050	+4 weeks
E_2	10	150	0	5690	37.5	5.25	0	0 weeks
			0.1	5520	36.6	5.20	572	+1 weeks
			1	2777	18.5	5.06	3207	+2 weeks
			10	1550	10.6	3.4	16360	+2 weeks
			100	0	0	0	22680	+4 weeks
E_3	30	150	0	5910	39.0	5.36	0	0 weeks
			0.1	5760	38.2	5.33	606	+1 weeks
			1	1965	12.9	4.22	3475	+2 weeks
			10	1342	8.8	2.75	17940	+3 weeks
			100	0	0	0	28500	+4 weeks
E_4	30	250	0	9870	39.2	6.05	0	0 weeks
			0.1	8920	35.6	5.20	1112	+1 weeks
			1	5680	22.8	5.59	30520	+3 weeks
			10	2870	11.2	3.65	72950	+4 weeks
			100	0	0	0	251280	+7 weeks

SCM problem as follows.

$$\text{minimize} \quad C \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{J(i)} \sum_{t=2}^{T} |r_{i,j,k,t} - r_{i,j,k,t-1}| + \sum_{i=1}^{N} \sum_{t=1}^{T} C_i^p(t) + \sum_{i=1}^{N} \sum_{j=1}^{J(i)} \sum_{t=1}^{T} C_{i,j}^a(t) \quad (39)$$

This optimization is subject to start and finish times, resource upper-bounds and activity precedence constraints, as described in Section 3.3. We consider each engineer as a resource type; thus, $R_{k,t} = 1, \forall t$. In consecutive timesteps $t-1$ and t, the engineer is either working on an activity $a_{i,j}$ of Project P_i, or working on another activity of the same project or a different project. When the engineer is working on the same activity, there is no switching between consecutive timesteps and the absolute difference of $|r_{i,j,k,t} - r_{i,j,k,t-1}|$ is zero. However, when the engineer works on a different activity or project, then the absolute difference is one and is multiplied by the fixed cost C of switching activities or projects. The total number of switches made by the engineer is multiplied by C in the objective function to obtain the total cost of switching. Our objective is to minimize the cost of switching across all engineers.

To verify this formulation, we use our generator described in Section 4.4 and create four input instances $E_{1,2,3,4}$ with 5, 10, 30, and 30 projects, respectively. Each instance has one activity per project (i.e., $J(i) = 1, \forall i = 1, \ldots, N$), and the number of engineers varying between 100 and 250 (i.e., $K = 100$, $K = 150$, or $K = 250$), $T = 90$ weeks, $R_{k,t} = U_{k,t} = 1, \forall t$, and $s(i, 1) = 1$. In instance E_1, we assess solutions with C set to 0, 0.1, 1, 10, and 100. $C = 0$ corresponds to zero cost of switching between projects, $C = 1$ corresponds to a small cost of switching, and $C = 100$ corresponds to a large cost of switching. Table VIII summarizes the number of switches made by each engineer over all projects. In instance E_1, when the cost of switching is zero, there are a total of 4108 switches without any impact to the overall schedule makespan. When the cost of switching is large ($C = 100$), there are zero switches but the overall schedule makespan increases by 4wk. When the cost of switching is small ($C = 1$), the total number of switches reduces from 4108 to 2180, but the makespan increases by 1wk and the total cost increases from 0 to 15050. Thus, we observe sensible behavior of the

trade-off between total number of switches (stable assignment) and overall schedule makespan. Instances E_2 and E_3 further show the trade-off in total number of switches and overall schedule makespan using different values of C for the same number of projects and engineers. Instance E_4 demonstrates scalability as the number of projects is increased to 30 and the number of engineers is increased to 250. In all instances, we reduce the total number of switches made by engineers by increasing C; however, the total cost increases as schedules of projects are pushed out by 1wk to 7wk.[20]

We also solve a variant of the engineer allocation problem by adding constraints that upper-bound the number of switches of each engineer during the overall schedule makespan. We use the same objective function as the SCM problem with additional constraints. We validate our solver for this variant using instance E_4 described earlier, and by varying the upper bound on the number of switches allowed per engineer as $\{+\infty, 30, 20, 10, 5, 1\}$. When the upper bound is $+\infty$, the total cost is zero and matches the total cost when $C = 0$ for instance E_4 in Table VIII. When the upper bound is 30, the total cost is 3830; when the upper bound is 20, the total cost is 33150; when the upper bound is 10, the total cost is 97150; when the upper bound is 5, the total cost is 116650; and when the upper bound is 1, the total cost is 226250.

5. CONCLUSIONS AND FUTURE WORK

The lack of tools for management of semiconductor design resources (servers, tool licenses, engineering headcount) can impact a company's bottom line by many millions of dollars per year. In this work, we capture multi-project, multi-resource constrained project scheduling as SCM and RCM MILP formulations that are readily solvable using commercial engines such as CPLEX. The MILP solutions provide *optimal* scheduling and allocation solutions for complex multi-tapeout management scenarios in a large design center. Aspects of our formulation that are unique to semiconductor design, and that take our work beyond the earlier RCPSP formulation of Kolisch and Hartmann [1999], Kolisch et al. [1992], and Kolisch and Sprecher [1996], include multiple resource pools and co-constraints between resources of different types. We demonstrate the flexibility and value of our optimization in three scenarios taken from the recent history of Company X's design center. (1) We find an optimal schedule for three concurrent tapeouts when a late-breaking RTL change hits one of the projects, and save 1.4 work-weeks of schedule compared to the solution deployed by the company. This level of saving corresponds to 2.7% of annual labor and infrastructure costs and enhances market competitiveness. (2) We find an optimal schedule for 20+ projects, subject to datacenter capacity limits and a tethering constraint with respect to original forecast resource allocations. Our solution shows that a slight relaxation of the tethering constraint would allow committed projects to proceed within resource limits. Our solution meets the schedule with no additional servers. By contrast, in the absence of decision support tools, the company's solution entailed the purchase of hundreds of additional servers. (3) We find an optimal allocation of human resources for four projects and save up to 37% of a particular resource type relative to the solution adopted by the company. In a non-US location, this single-resource type reduction would imply a ∼$3.7M savings for the company within a half-year project scheduling makespan. We also provide "what-if" analyses capabilities with our solver and demonstrate sensitivity analyses (schedule benefits of incremental resources) and scalability of our solution approach. Since we introduce new concepts such as conditionally-shared, segregated,

[20]Instance E_1 has ∼46K variables, ∼350K constraints, and a runtime of around 3min. Instance E_2 has ∼150K variables, ∼1.1M constraints, and a runtime of around 28min. Instance E_3 has ∼450K variables, ∼3.2M constraints, and a runtime of around 50min. Instance E_4 has ∼680K variables, ∼5.4M constraints, and a runtime of around 2.3h.

and fully-shared resource types along with resource co-constraints, we are unable to compare against any previously existing MILP formulations of such problems. Using a (stochastic) instance generator, we demonstrate impact on makespan with stochasticity in resource requirements and iterations in activities.

Our work is applicable across multiple stages of project and capacity planning processes. For example, it can be used (i) as part of a fiscal planning process to comprehend the overall resource requirements of a site or a computing cluster, (ii) by program management as a what-if tool so that infrastructure can join engineering headcount as a factor in scheduling decisions, and (iii) by "Engineering Compute" operations teams to understand the impact of product roadmap or schedule changes on datacenter and EDA licensing infrastructure so that corrective actions may be taken in a proactive and principled manner. By providing a foundation for improved engineering resource allocation to maintain high overall resource utilizations and low schedule latencies, we enable design organizations to improve design throughput and efficiency with given resources. Ultimately, this helps to continue the scaling of design cost efficiencies that are so vital to the IC industry. Our solving of instances with co-constraints, stability constraints and tethering forecast resource allocations may be of interest in other application domains, such as industrial assembly (e.g., automobile assembly), air traffic management, and workflow scheduling in grid computing [Laborie and Godard 2007].

Our formulations can be improved in a number of directions that are the subject of ongoing investigation. For instance, it will be helpful to be able to automatically determine threshold values of inputs (e.g., the schedule length T or the tethering constraint δ) at which feasible solutions exist. Scheduling decisions should also comprehend distributions of job sizes or job complexities (which can vary per block and according to the state of a project), and automatic change of timestamps when schedule changes occur. Solutions can be further stabilized by adopting iterative optimization approaches. A further open direction is to optimize robustness of scheduling solutions in the face of stochasticity in resources and personnel.

APPENDIX

Table IX maps activities and resources from problem instances in Section 4.1 and Section 4.3 to chip design flow terminologies.

Table IX. Glossary for the Schedule Modification Use Case (Section 4.1) and (Human) Resource Allocation Use Case (Section 4.3)

Schedule modification use case (Section 4.1)		**Human resource allocation use case (Section 4.3)**	
Activity / Resource	Chip Design Flow Mapping	Activity / Resource	Chip Design Flow Mapping
A1	Placement	A12	Block-Level Design (BLD)
A2	Routing	A13	Full-Chip Design (FCD)
A3	Search and Repair	A14	Block-Level Verification (BLV)
A4	Extraction	A15	Full-Chip Verification (FCV)
A5	Static Timing Analysis (STA)	A16	Block-Level Physical Design (BLPD)
A6	Functional ECO	A17	Full-Chip Physical Design (FCPD)
A7	Extraction	A18	Gate-Level Simulation (GLS)
A8	STA (per corner)	A19	Emulation (EMU)
A9	Timing ECO	HR1	Design Resources
A10	Extraction	HR2	Design Verification Resources
A11	STA (per corner)	HR3	PD Resources
L1	P&R License	HR4	EMU Resources
L2	RCX License	–	–
L3	STA License	–	–

REFERENCES

P. Agrawal, B. Chatterjee, A. B. Kahng, P. K. Myana, and S. Nath. 2015. Optimal multi-tapeout project scheduling for enterprise-scale design management and cost reduction. *Work-in-Progress, DAC*.

M. Ayala and C. Artigues. 2010. *On Integer Linear Programming Formulations for the Resource-Constrained Modulo Scheduling Problem, Rapport LAAS*. Technical Report 10393.

P. Baptiste and S. Demassey. 2004. Tight LP bounds for resource constrained project scheduling. *Operations Research Spectrum* 26, 251–262.

D. Bienstock and M. Zuckerberg. 2009. A new LP algorithm for precedence constrained production scheduling. *Optimization Online* 1–33.

A. Bonfietti, M. Lombardi, L. Benini, and M. Milano. 2014. Cross cyclic resource-constrained scheduling solver. *Artificial Intelligence* 206, 25–52.

W.-T. J. Chan, A. B. Kahng, S. Nath, and I. Yamamoto. 2014. The ITRS MPU and SOC system drivers: Calibration and implications for design-based equivalent scaling in the roadmap. In *Proceedings of the IEEE International Conference on Computer Design*. 153–160.

N. Christofides, R. Alvarez-Valdes, and J. M. Tamarit. 1987. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research* 29, 262–273.

S. Fenstermaker, D. George, A. B. Kahng, S. Mantik, and B. Thielges. 2000. METRICS: A system architecture for design process optimization. In *Proceedings of the ACM/IEEE Design Automation Conference*. 705–710.

R. Friese, T. Brinks, C. Oliver, H. J. Siegel, and A. A. Maciejewski. 2012. Analyzing the trade-offs between minimizing makespan and minimizing energy consumption in a heterogeneous resource allocation problem. In *Proceedings of International Conference on Advanced Communications and Computation*. 81–89.

A. B. Kahng and G. Smith. 2002. A new design cost model for the 2001 ITRS. In *Proceedings of International Symposium on Quality Electronic Design*. 190–193.

B. Keller and G. Bayraksan. 2009. Scheduling jobs sharing multiple resources under uncertainty: A stochastic programming approach. *IIE Transactions* 42, 16–30.

R. Kolisch and S. Hartmann. 1999. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. *International Series in Operations Research & Management Science* 14, 147–178.

R. Kolisch and A. Sprecher. 1996. PSPLIB – A project scheduling problem library. *European Journal of Operational Research* 96, 205–216.

R. Kolisch, A. Sprecher, and A. Drexl. 1992. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science* 41, 10, 1693–1703.

B. A. Kramer and C. L. Hwang. 1991. Resource constrained project scheduling: modeling with multiple alternatives. *Mathematical and Computational Modeling* 15, 8, 49–63.

P. Laborie and D. Godard. 2007. Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Multidisciplinary International Scheduling Conference*, 276–284.

M. Li, Y. Zhang, W. Jiang, and J. Xie. 2009. A particle swarm optimization algorithm with crossover for resource constrained project scheduling problem. In *Proceedings of International Conference on Services Science, Management and Engineering*. 69–72.

Y. Li, J. Han, and W. Zhou. 2014. Cress: Dynamic scheduling for resource constrained jobs. In *Proceedings of International Conference on Computational Science and Engineering*. 1945–1952.

S. Mohanty and M. K. Nayak. 2011. Optimization model in human resource management for job allocation in ICT project. *International Journal of the Computer, the Internet and Management* 19, 3, 21–27.

R. H. Mohring, A. S. Schulz, F. Stork, and M. Uetz. 2001. On project scheduling with irregular starting time costs. *Operations Research Letters* 28, 149–154.

Qualcomm Inc. (IT project manager). 2014. Personal Communication.

Z. Qiong, G. Yichao, Z. Ging, Z. Jie, and C. Xuefang. 2010. An ant colony optimization model for parallel machine scheduling with human resource constraints. In *Proceedings of International Conference on Digital Enterprise Technology Advances in Intelligent and Soft Computing*, Vol. 66. 917–926.

F. Salewski, A. Schirmer, and A. Drexl. 1997. Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. *European Journal of Operational Research* 102, 1, 88–110.

G. Smith. 2014. Personal Communication.

G. Smith. 2014. Updates of the ITRS design cost and power models. In *Proceedings of the IEEE International Conference on Computer Design*. 161–165.

F. B. Talbot. 1982. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science* 28, 10, 1197–1210.

Cadence Assura QRC. Retrieved July 04, 2016 from https://www.cadence.com/content/cadence-www/global/en US/home/tools/digital-design-and-signoff.html.

Cadence Innovus Implementation System. Retrieved July 4, 2016 from https://www.cadence.com/content/cadence-www/global/en US/home/tools/digital-design-and-signoff.html.

Cadence Tempus Timing Signoff. Retrieved July 4, 2016 from https://www.cadence.com/content/cadence-www/global/en US/home/tools/digital-design-and-signoff.html.

Dassault Systems Enovia Synchronicity. Retrieved July 8, 2016 from http://www.3ds.com/products-services/enovia/products/v6/synchronicity-designsync/.

Glassdoor. Retrieved January 1, 2017 from https://www.glassdoor.com/Salaries/index.htm.

How Green is my Silicon Valley? Retrieved July 4, 2016 from http://dac.com/sites/default/files/DACArchive/pubs/46DACFinal Prgm.pdf.

IBM ILOG CPLEX. Retrieved July 4, 2016 from http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/.

IC Manage. Retrieved July 8, 2016 from https://www.icmanage.com/ic-design-management-best-practices/.

inMotion Creative Project Management. Retrieved July 4, 2016 from http://explore.inmotionnow.com/capterra-project-management.

ITRS. Retrieved July 4, 2016 from http://www.itrs2.net/.

Nefelus Design Tools. Retrieved July 8, 2016 from http://www.nefelus.com/design-tools/.

UCSD Design Cost Optimization Solver for Multi-Tapeout Project Scheduling. Retrieved July 4, 2016 from http://vlsicad.ucsd.edu/MILP/.

When The Chips are Down. Retrieved July 4, 2016 from http://qz.com/387490/as-moores-law-turns-50-computer-chips-continue-to-get-cheaper-and-more-powerful/.

Platform Load Sharing Facility. Retrieved July 4, 2016 from http://www-03.ibm.com/systems/services/platformcomputing/lsf.html.

PSLIB Data Sets. Retrieved July 4, 2016 from http://www.om-db.wi.tum.de/psplib/download.html.

Runtime Design Automation. Retrieved July 4, 2016 from http://www.rtda.com/.

Qualcomm Snapdragon. Retrieved July 4, 2016 from https://www.qualcomm.com/products/snapdragon.

Salesforce Project Management. Retrieved July 8, 2016 from https://www.salesforce.com/.

Samsung Exynos. Retrieved July 4, 2016 from http://www.samsung.com/semiconductor/products/exynos-solution/application-processor/.

Synopsys IC Compiler. Retrieved July 4, 2016 from http://www.synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/default.aspx.

Synopsys PrimeTime. Retrieved July 4, 2016 from http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx.

Synopsys Star-RCXT. Retrieved July 4, 2016 from http://www.synopsys.com/Tools/Implementation/SignOff/Pages/StarRC-ds.aspx.

Received July 2016; revised November 2016; accepted December 2016