Bi-Objective Optimization for Scheduling in Heterogeneous Computing Systems

Tony Maciejewski,
Kyle Tarplee, Ryan Friese, and Howard Jay Siegel

Department of Electrical and Computer Engineering

Colorado State University
Fort Collins, Colorado, USA

Outline

- motivation and environment
- bi-objective optimization
- experimental setup and results
- summary

Motivation: Energy-Awareness

- energy consumption on high performance computing systems – very expensive
- in general there is a correlation between increasing compute power and increasing energy consumption
- how can we maintain or increase performance while using less energy?
  - energy-aware resource allocation

faster performance

greater energy requirements

billions spent in electricity costs

Colorado State University

Heterogeneous Parallel Computing System

- interconnected set of different types of machines with varied computational capabilities

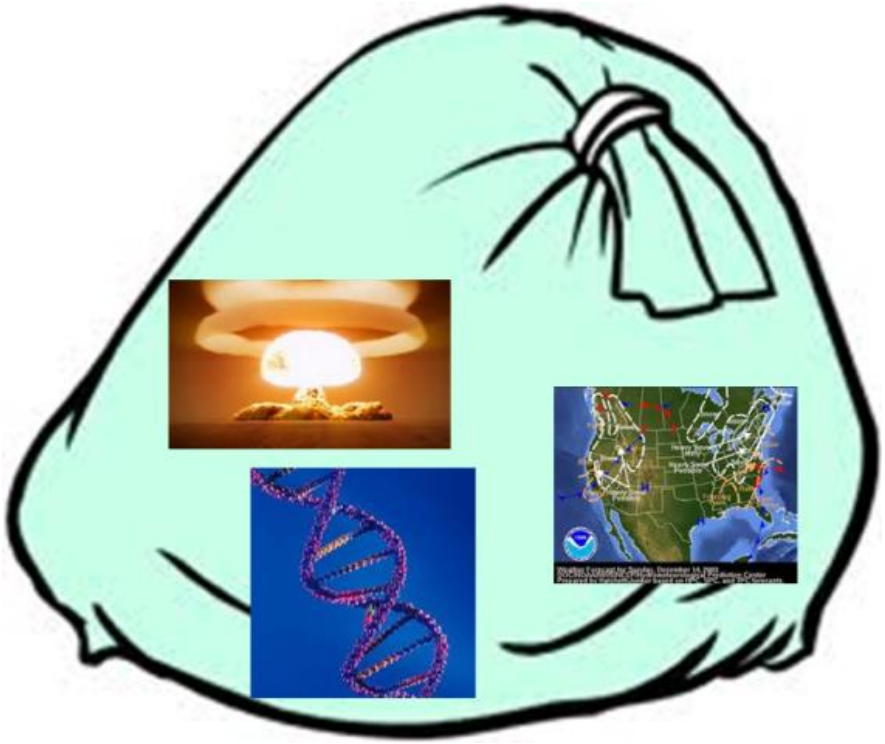
- workload of tasks with different computational requirements
- each task may perform differently on each machine
  - furthermore: machine A can be better than machine B for task 1 but not for task 2

- **resource allocation**:
  assign (map) tasks to machines
  to optimize some performance measure
  - NP-complete (cannot find optimal in reasonable time)
  - ex.: 5 machines and 30 tasks → 5^{30} possible assignments
    - 5^{30} nanoseconds > 1,000 years!
  - use heuristics to find near optimal allocation

Colorado State University

Environment

- static and offline environment
  - bag of tasks (batch)
  - every task in the workload is known *a priori*



Task and Machine Types

- task type – similar computational requirements
- machine type – similar performance capabilities
- we use an **Estimated Time to Compute (ETC)** matrix
  - gives estimated time for executing each task type on each machine type
- we use an **Average Power Consumption (APC)** matrix
  - gives average power consumed for executing each task type on each machine type
- in real world: use historical data, experiments, benchmarks
- simulator uses a synthetic workload extrapolated from real data found on openbenchmarking.org

Calculating Energy Consumption

- recall: Estimate Time to Compute (ETC) - the execution time of a given task on a given machine
- recall: Average Power Consumption (APC) - the average power consumption of a given task on a given machine

ETC values (seconds)

	M1	M2
T1	8	10
T2	9	12
T3	7	11

APC values (watts)

	M1	M2
T1	115	95
T2	105	87
T3	125	90

energy of T3 on M2 = 11 seconds * 90 Watts = 990 Joules

Environment Considered

- tasks are assumed to be **independent**
  - communication is not required between the tasks and there are no precedence constraints
- tasks are assumed to be **serial**
  - execute on a single machine
- **dedicated** environment
  - system executes a single bag of tasks

Outline

- motivation and environment
- bi-objective optimization
- experimental setup and results
- summary

Analyzing Resource Allocations

- resource allocation
  - exploit the heterogeneous nature of the system to optimize some objective
- we are considering two objectives that typically conflict
  - **makespan**
    - total amount of time it takes for all the tasks in the batch to finish executing across all machines
  - **energy consumption**
    - total amount of energy consumed to execute all tasks within the batch

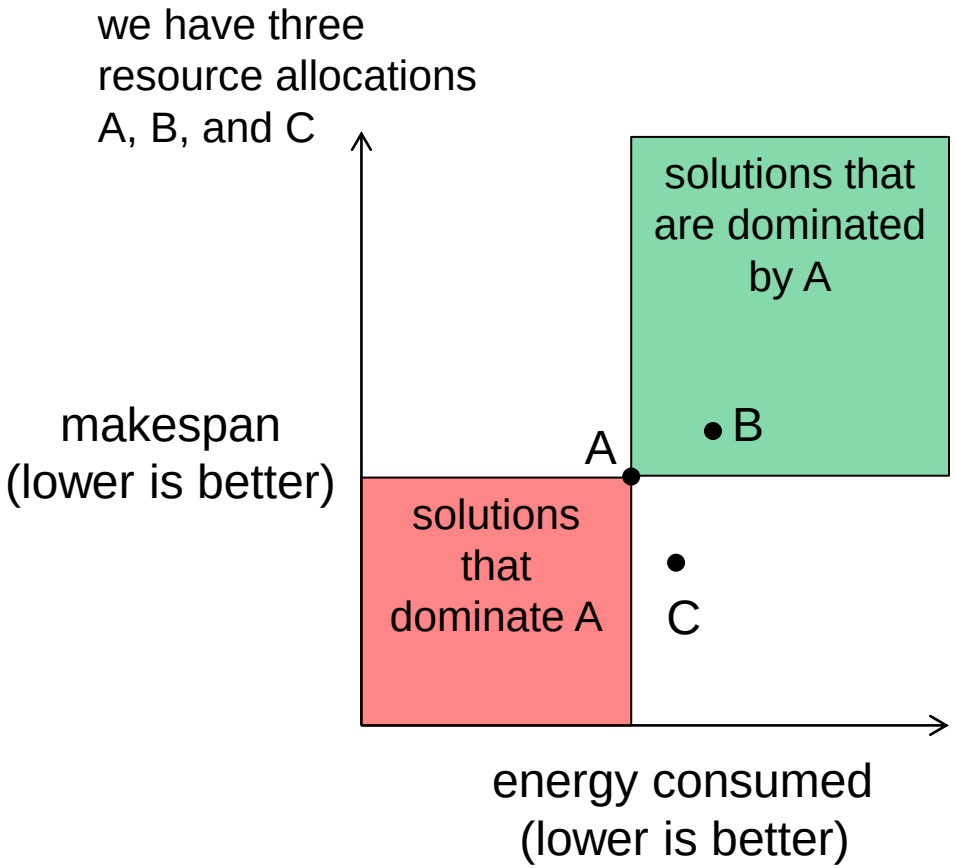
goal:

- evaluate trade-offs between makespan and energy consumption

Bi-Objective Optimization for Energy-Aware Schedules

- why is it important to study bi-objective optimization?
  - real world problems often have multiple objectives
  - objectives may conflict with each other
- used to understand how different resource allocations trade-off between:
  - system makespan
  - energy consumed
- allows system administrators to gain insights on how their systems operate

Pareto Fronts



- B is dominated by A
  - A uses less energy while having a lower makespan
  - A is the better solution
- neither A nor C dominate each other
  - A is better for energy
  - C is better for makespan
  - neither is better in both

- a Pareto front contains all the solutions which are not "dominated" by any other solution
- Pareto fronts facilitate trade-off analysis

Outline

- motivation and environment
- bi-objective optimization
- experimental setup and results: a genetic algorithm
  - approximate optimal solutions to the exact problem
- summary

Bi-Objective Genetic Algorithm

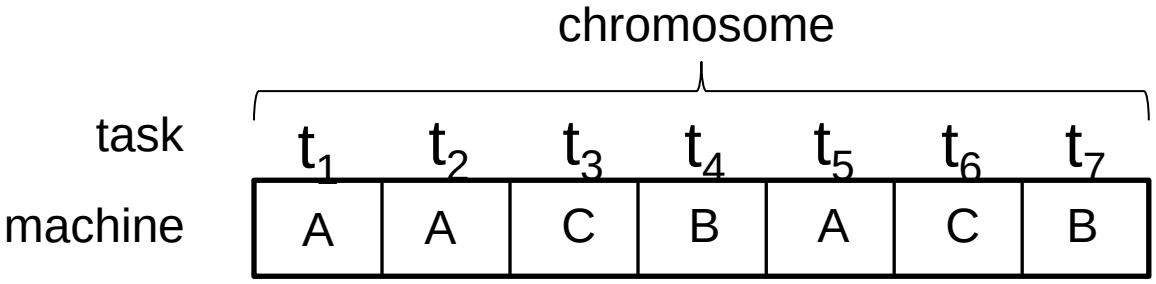
- genetic algorithms are search heuristics used to find approximate solutions to optimization problems
- solutions evolve over time by passing on useful traits
- given enough time, the solutions will converge towards the set of optimal solutions
- preferably solutions should be diverse and evenly distributed across the Pareto front

Genetic Algorithm for Energy-Aware Scheduling

- we are adapting a genetic algorithm from literature for use within our environment
  - "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," Deb et al., IEEE Transactions on Evolutionary Computation, 2002
- a solution in our environment is a complete resource allocation, i.e., mapping of tasks to machines
- NSGA-II builds Pareto front with solutions that are diverse and evenly distributed
- to use with our environment, we had to design:
  - chromosome structure
  - crossover operation
  - mutation operation

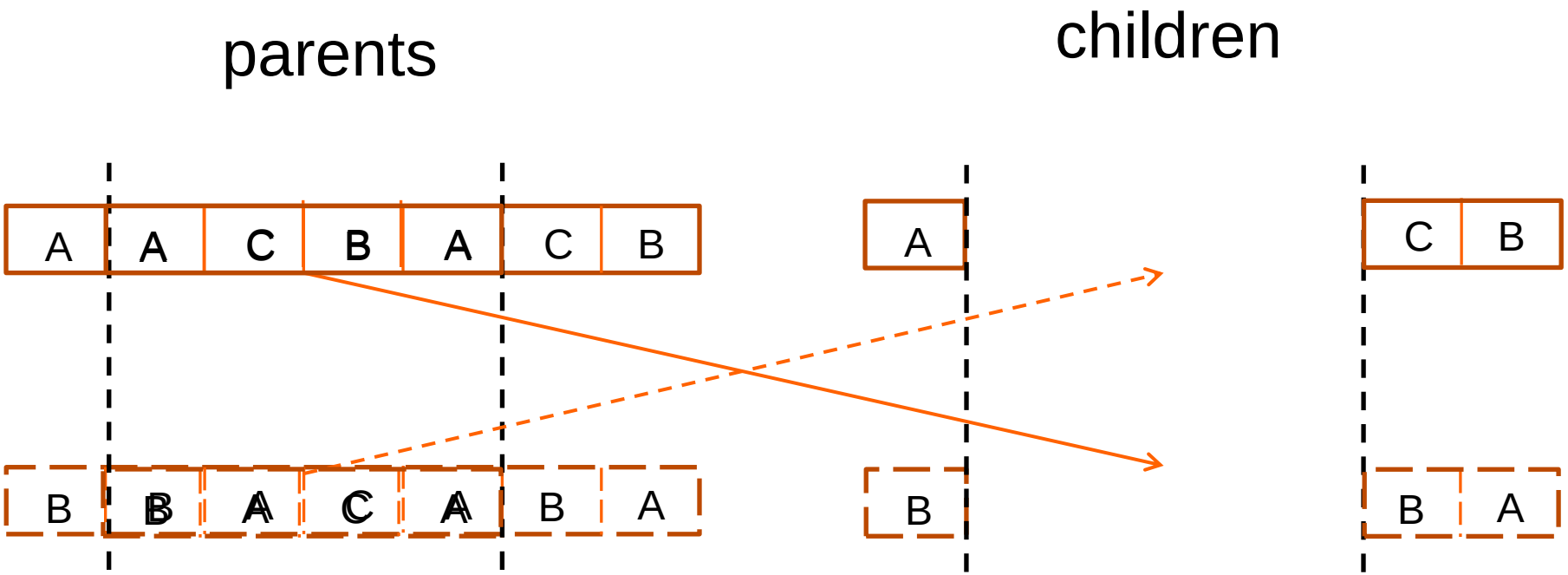
Chromosome Structure

- a chromosome represents a complete resource allocation for mapping each task to a machine (solution)
- the i^{th} entry of the chromosome is the machine task *i* is assigned to



Crossover Operation

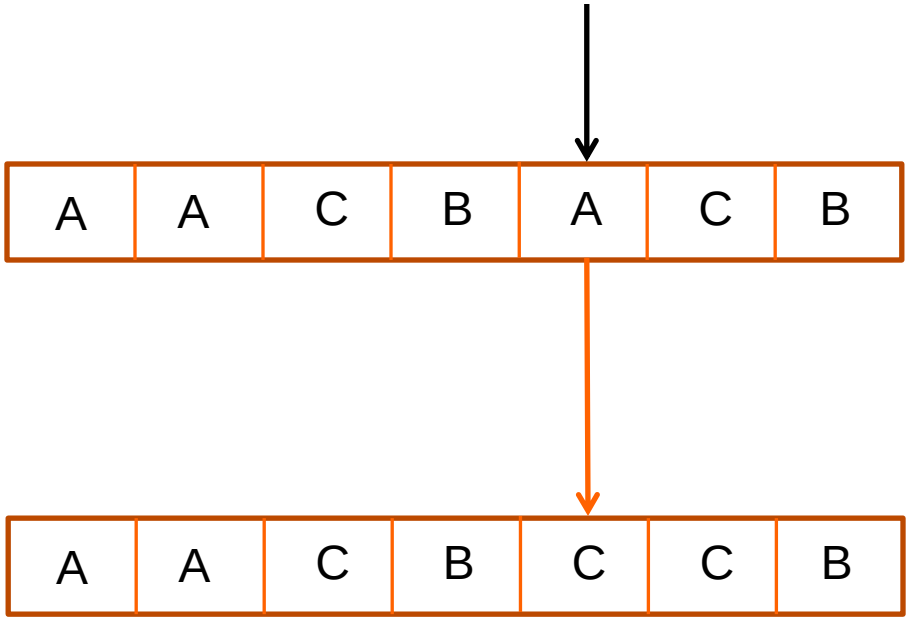
- select two random chromosomes
- find two random cuts within chromosomes
- swaps machines between these two cuts



Colorado State University

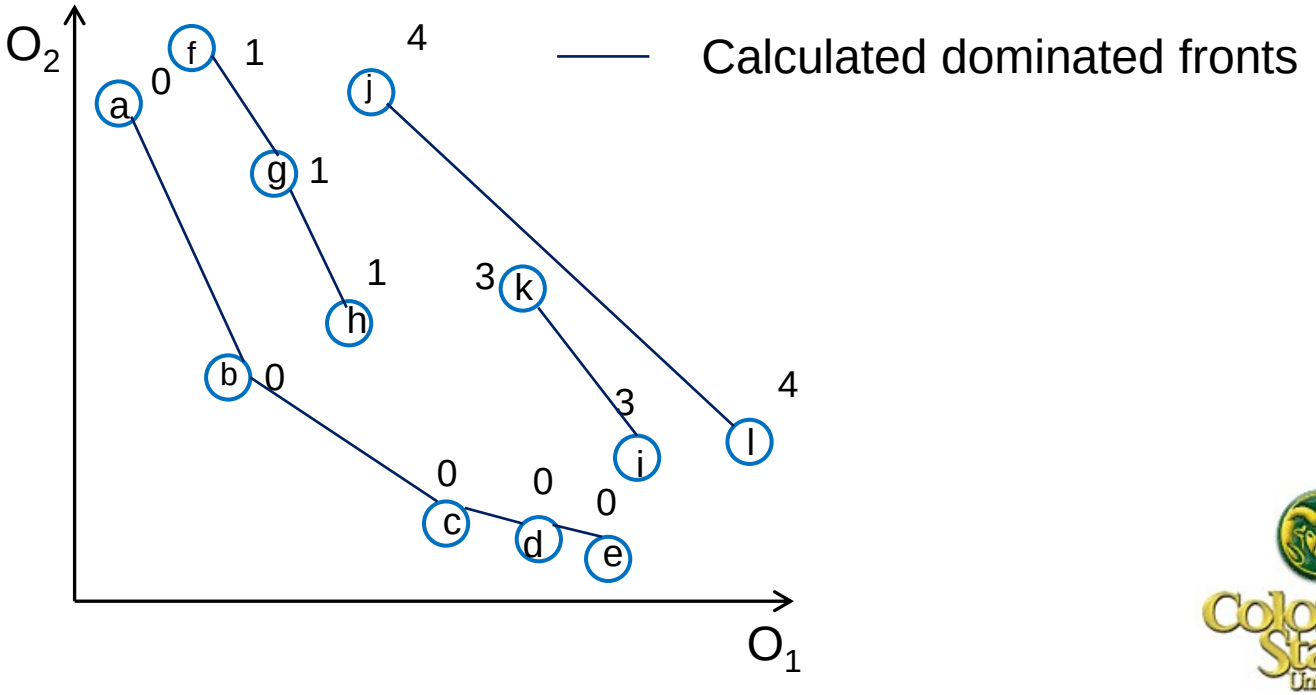
Mutation Operation

- select a random task
- replace current machine with a new random machine



NSGA-II: Fitness Function

- based on Pareto dominance
- sorts solutions into domination fronts
  - the points in a front are determined by the number of solutions which dominate it
  - this allows densely populated regions to be penalized



NSGA-II: Diversity and Elitism

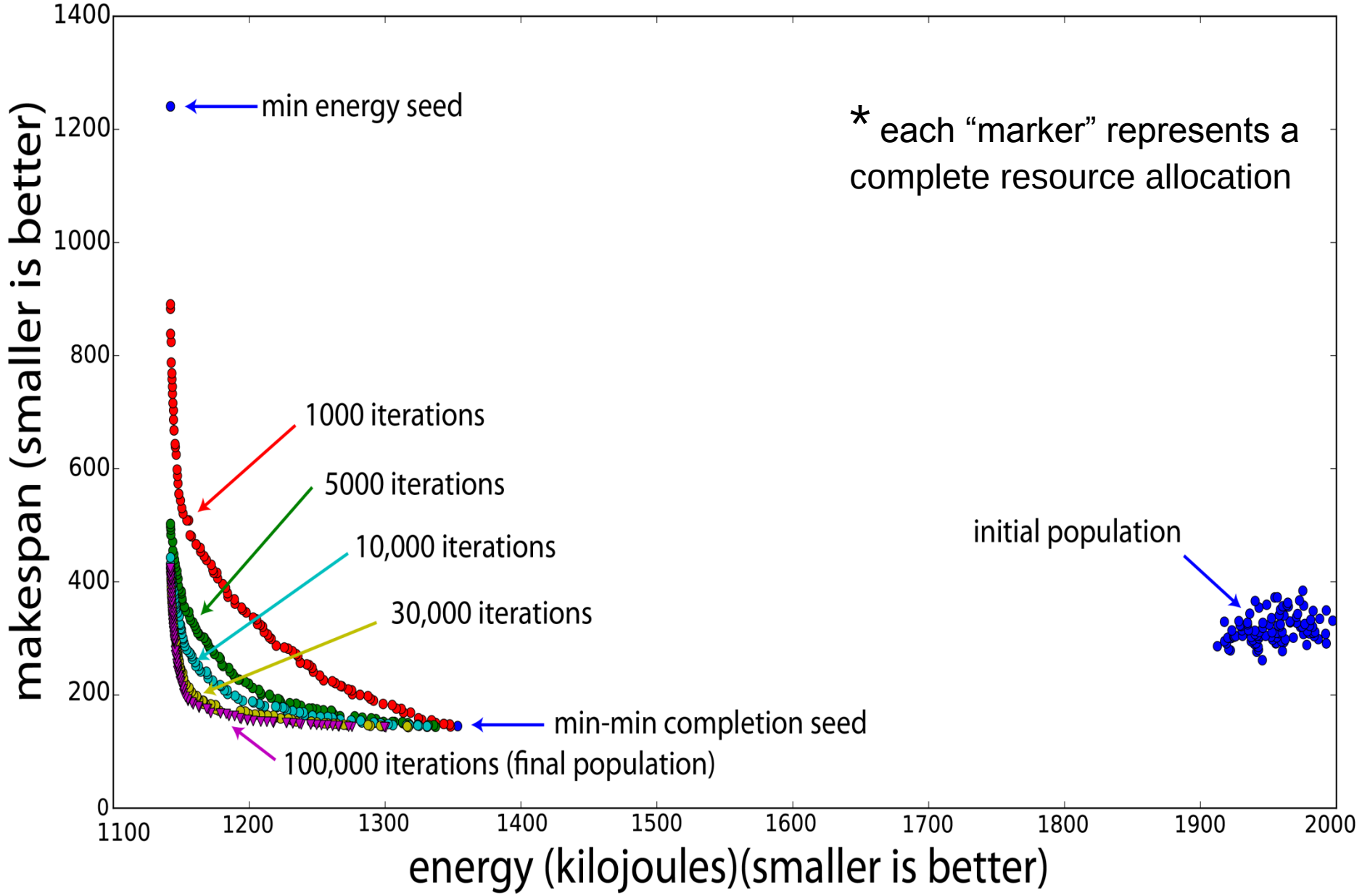
- for each equally dominated front
  - calculate the crowding distance for each solution
    - for each objective function F_k, sort solutions of the front in ascending order.
    - $cd(x) = \sum_k \frac{F_k(x_{[i+1]}) - F_k(x_{[i-1]})}{F_k^{max}(x) - F_k^{min}(x)}; i = 1, \ldots, |P_j| - 2$
    - set the crowding distance for the endpoints to infinity in order to make sure they are preserved from generation to generation

- select next population by picking solutions with
  - lower number of dominated solutions
    - bigger crowding distance

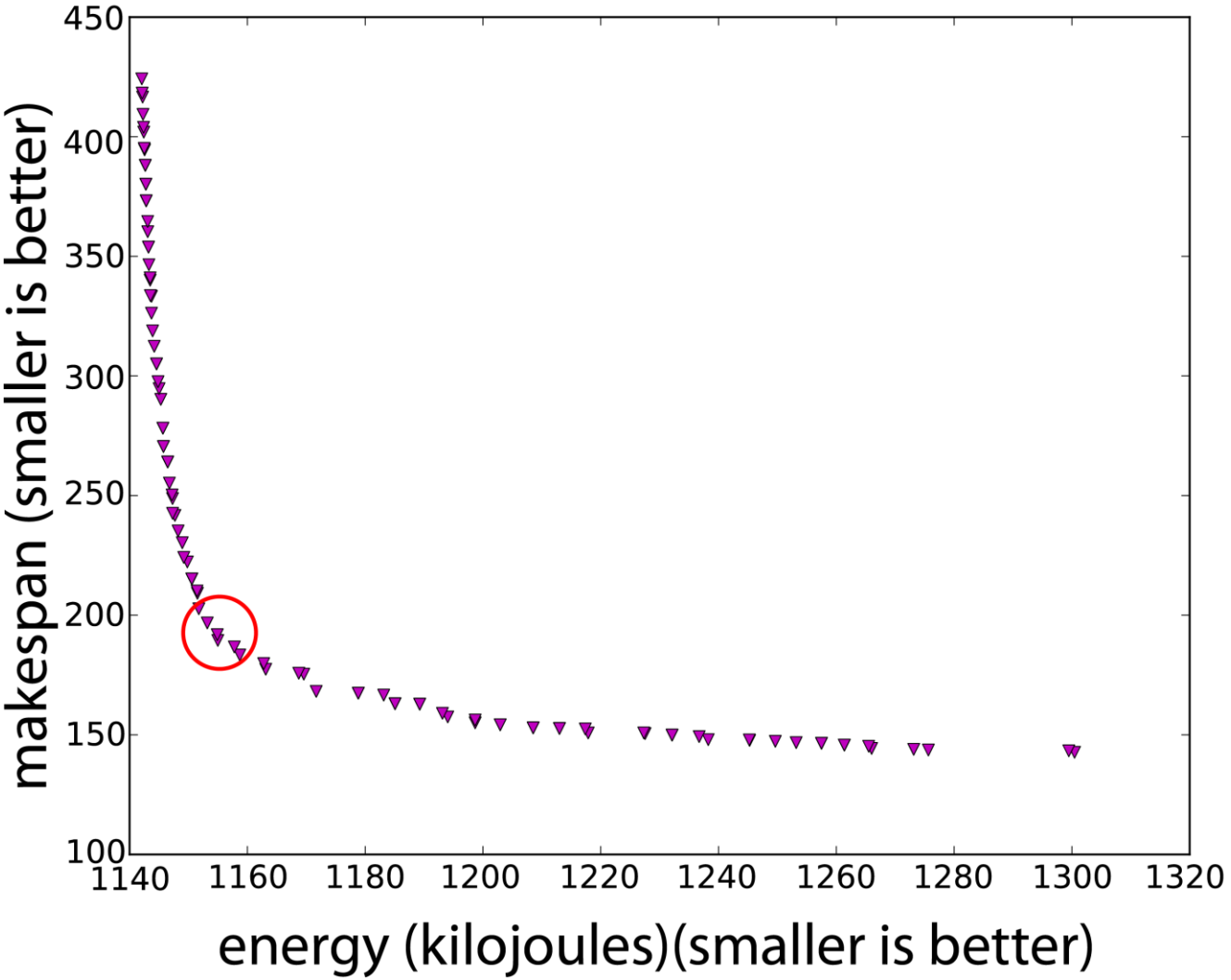
Seeding Heuristics for Initial Population

- one chromosome is the Min Energy seed
  - Min Energy
    - map task to machine that consumes the least energy (energy = execution time * power consumption)
- one chromosome is the Min-Min completion time seed
  - Min-Min Completion Time
    - two-stage heuristic assigning tasks to machines, picking assignments with least completion time (completion time = task start time + execution time)
- the rest of the chromosomes are created randomly

Evolution of Pareto Front Through 100,000 iterations

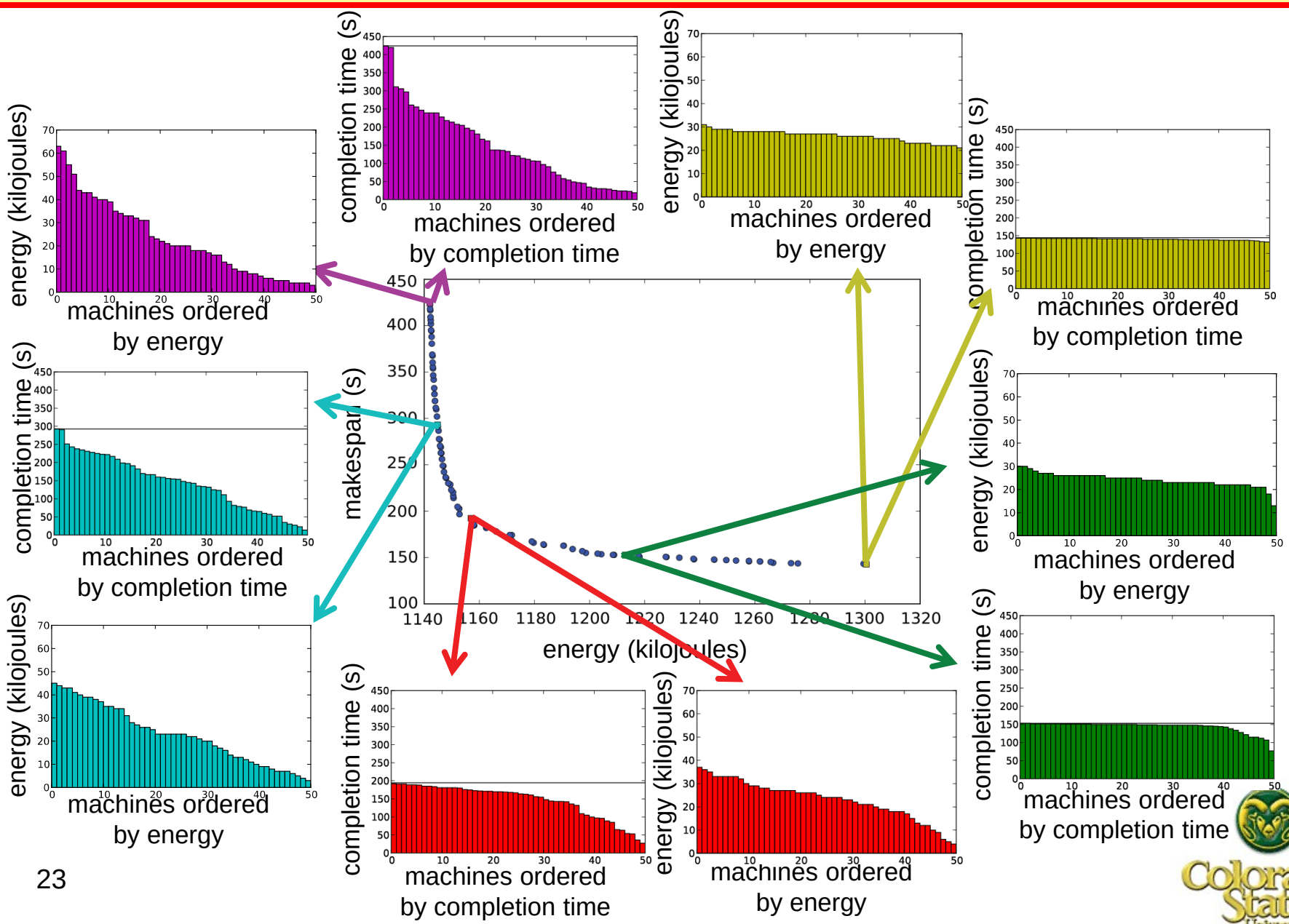


Final Pareto Front



Colorado State University

Analysis of Five Solutions from the Final Pareto Front

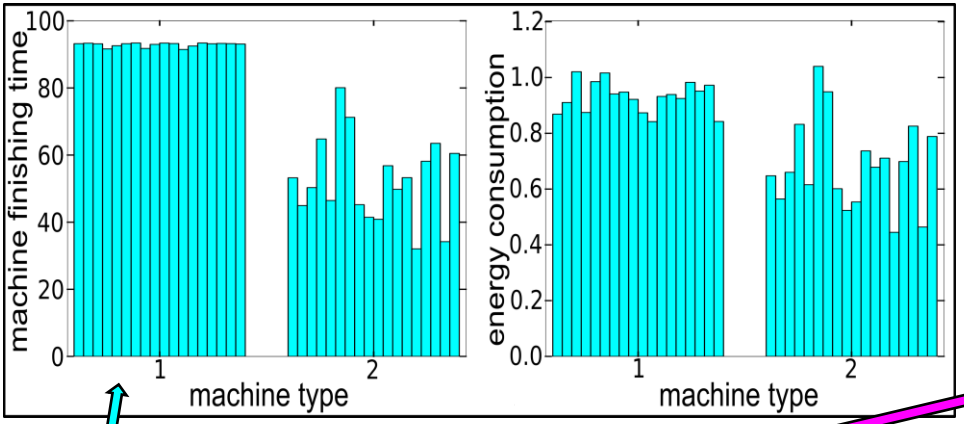


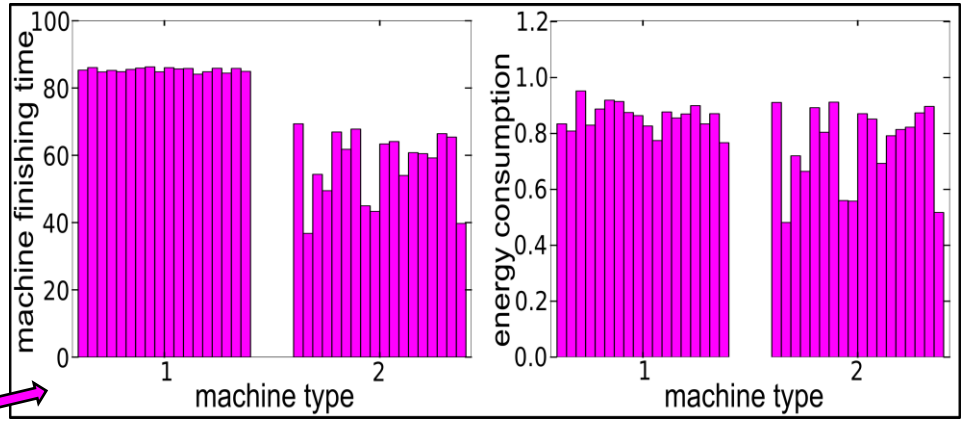
Colorado State University

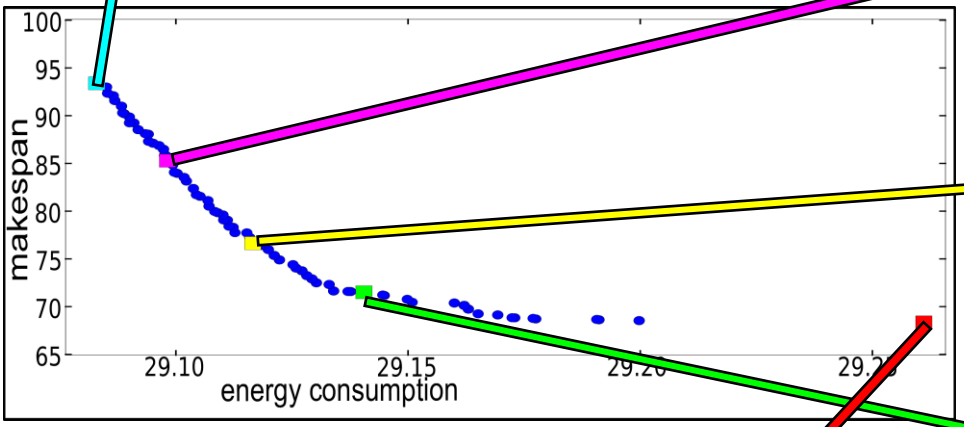
Variations in Computing Environment

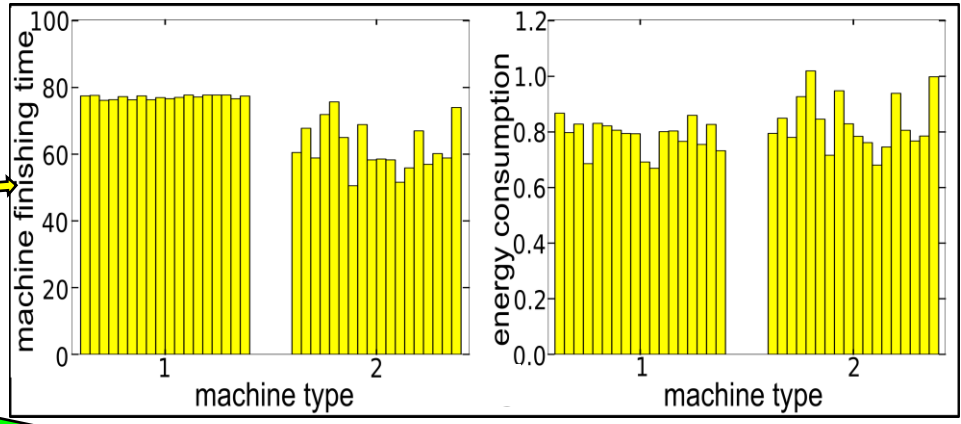
- to illustrate the versatility of our approach we modeled and simulated three computing environments
- 36 machines
  - 2 machine types (18 machines per type)
  - 6 machine types (6 machines per type)
  - 9 machine types (4 machines per type)
- 9 machine types are based on real machines
  - the 2 and 6 machine types are subsets of the 9 machine types
- 30 task types
  - 1000 tasks
- Pareto fronts were generated using a bi-objective genetic algorithm
- any algorithm that creates Pareto fronts could be used as well

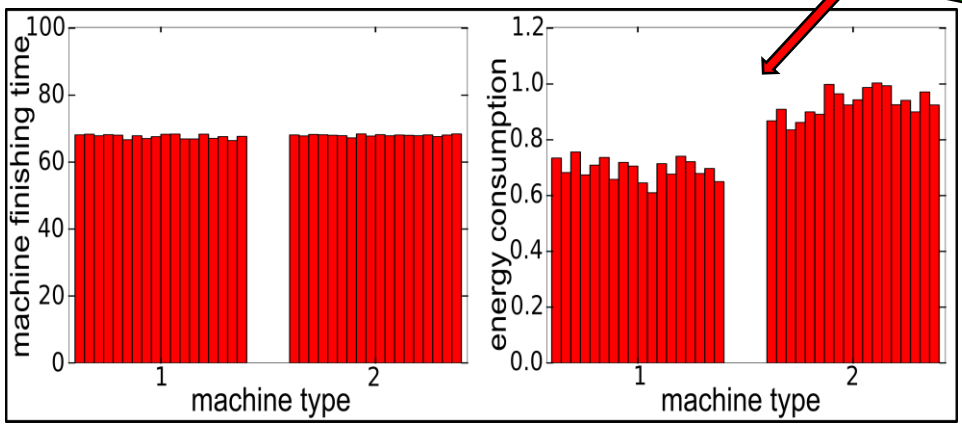
2 Machine Type Environment Analysis

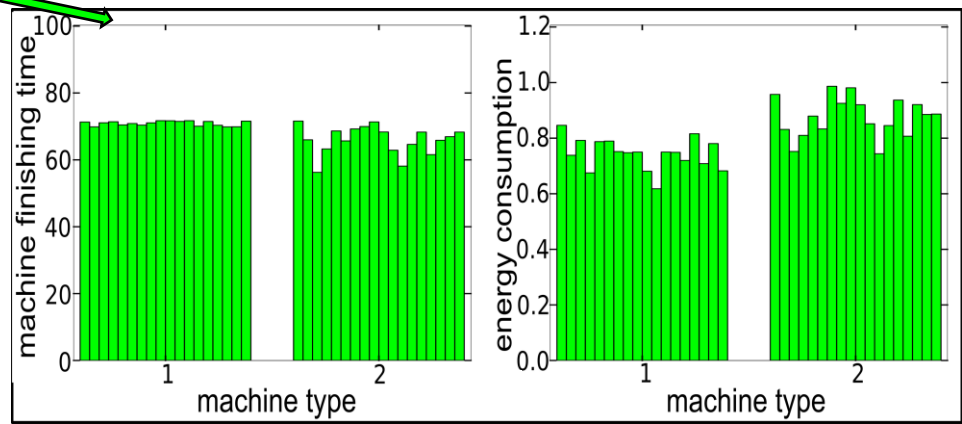




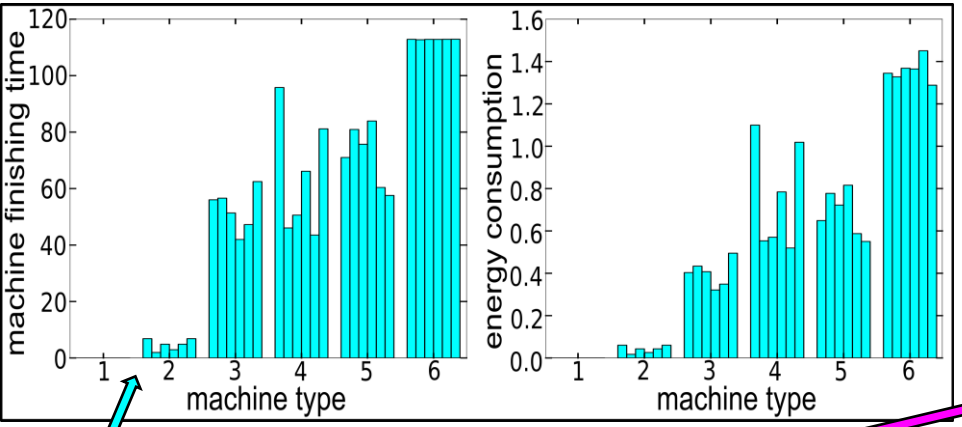


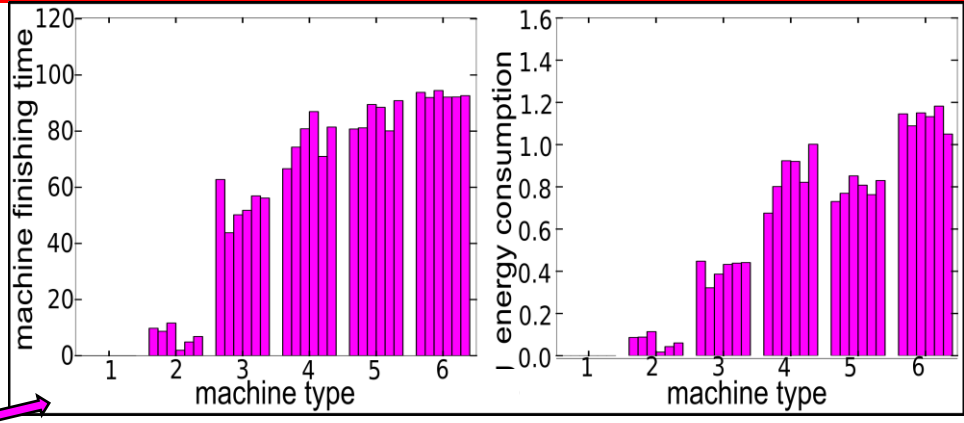


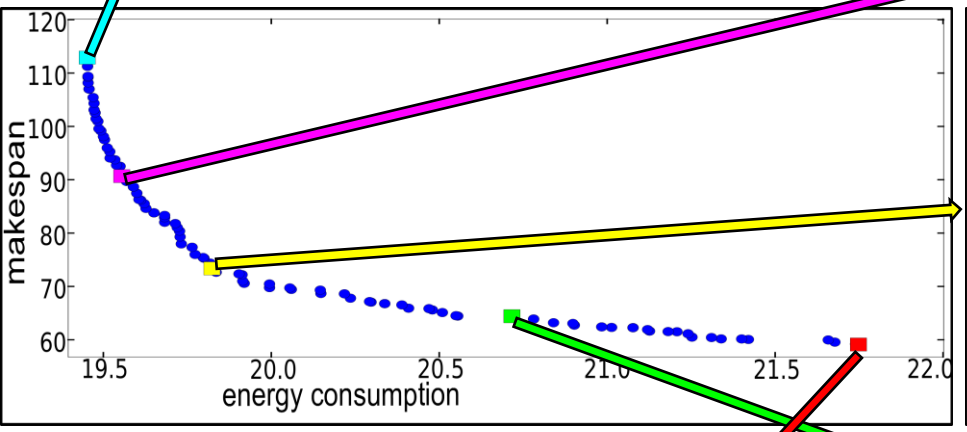


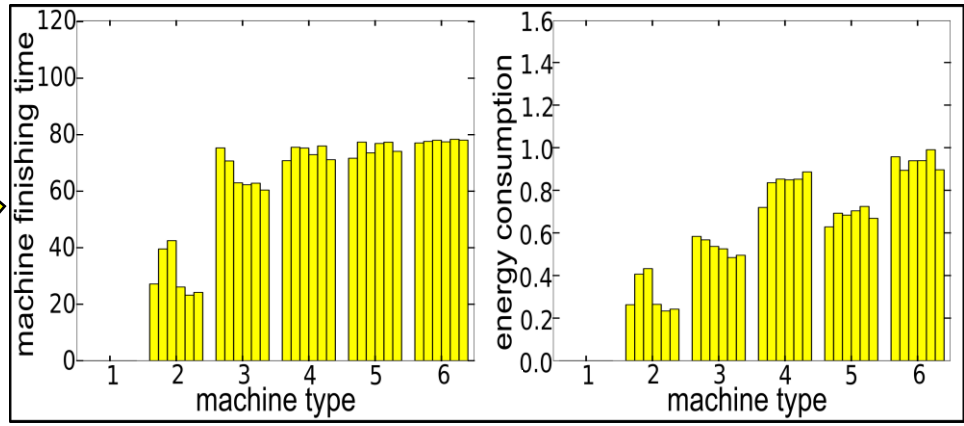


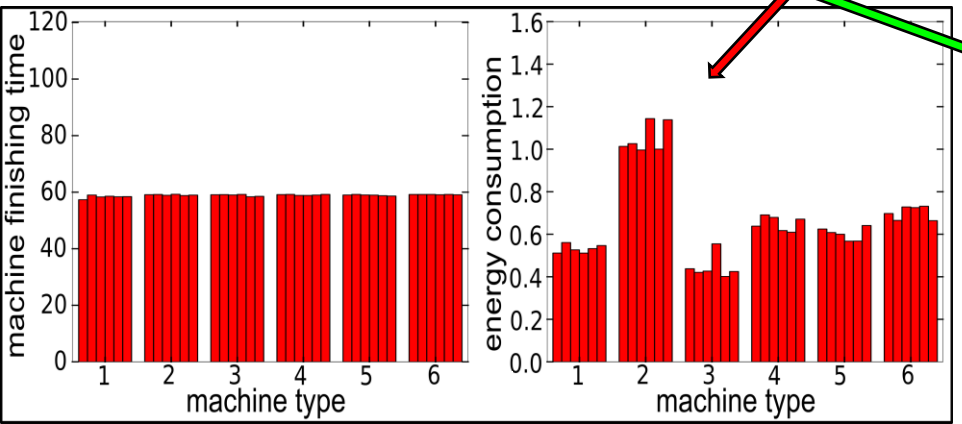
6 Machine Type Environment Analysis

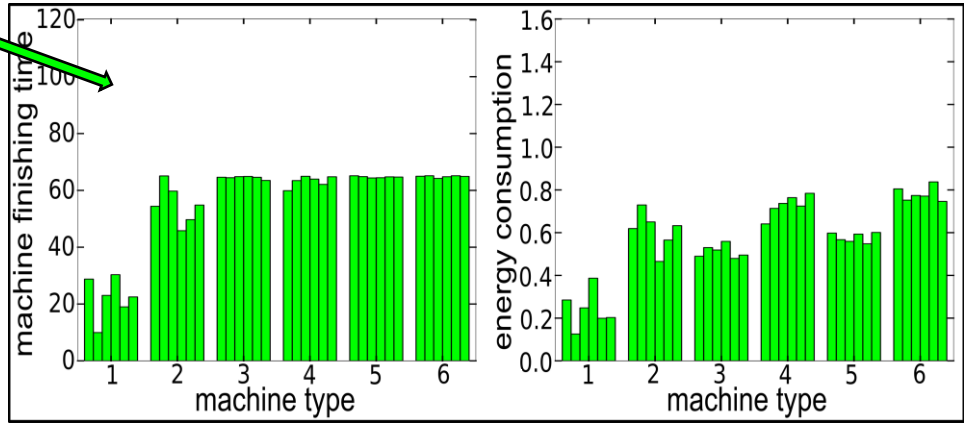




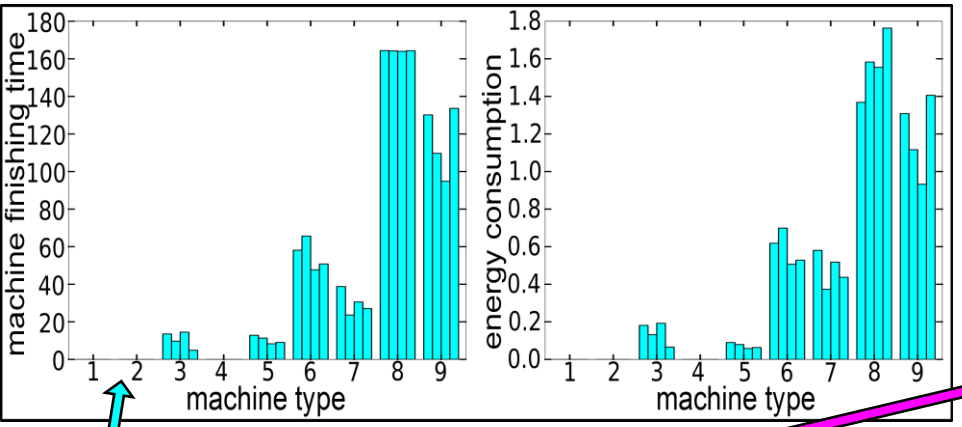


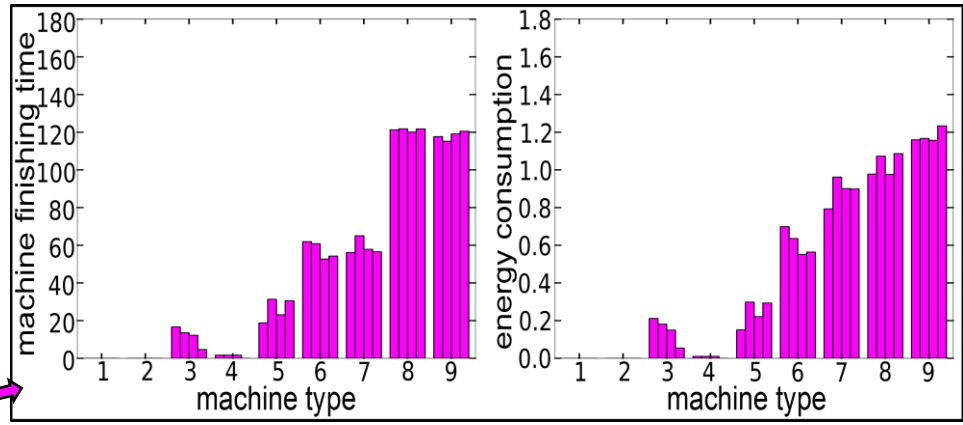


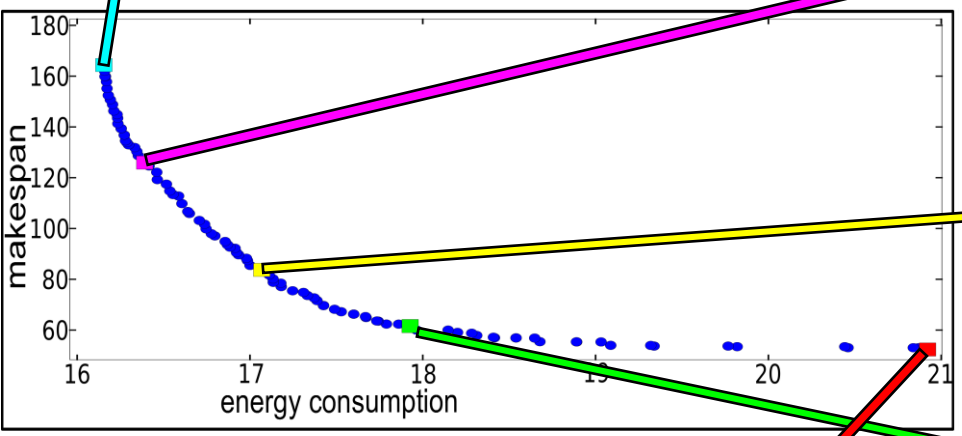


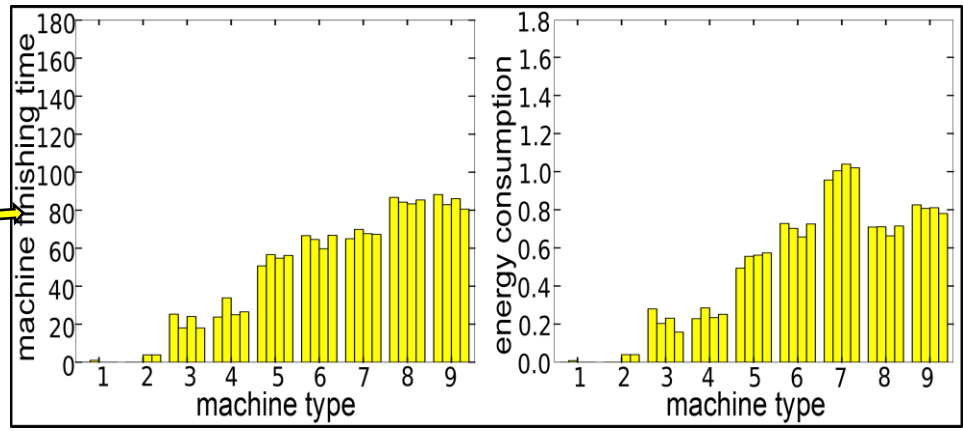


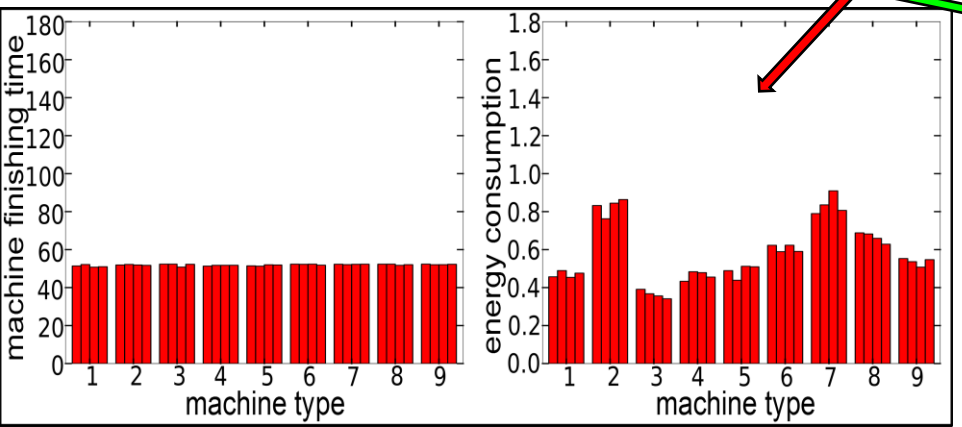
9 Machine Type Environment Analysis

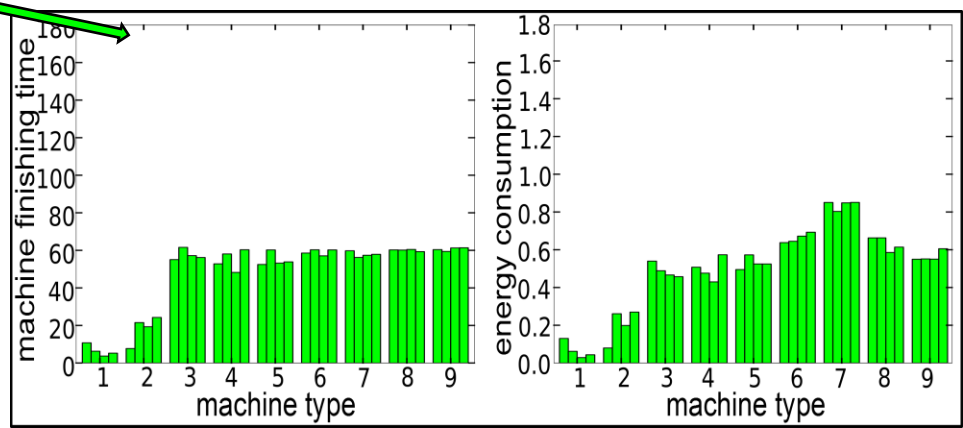












Outline

- motivation and environment
- bi-objective optimization
- experimental setup and results: linear programming
  - optimal solutions to an approximate problem
- summary

Recall Our Problem Statement

- static scheduling
  - single bag-of-tasks
  - task assigned to only one machine
  - machine runs one task at a time
- heterogeneous tasks and machines
- desire to reduce energy consumption (operating cost) and makespan
- to aid decision makers
  - find high-quality schedules for both energy and makespan
  - desire computationally efficient algorithms to compute Pareto fronts

Preliminaries

- simplifying approx: **tasks are divisible among machines**
- T_i − number of tasks of type *i*
- M_j − number of machines of type *j*
- x_{ij} − number of tasks of type *i* assigned to machine type *j*
- ETC_{ij} − estimated time to compute for a task of type *i* running on a machine of type *j*
- finishing time of machine type *j* (lower bound):

$$F_j = \frac{1}{M_j} \sum_i x_{ij} ETC_{ij}$$

- APC_{ij} − average power consumption for a task of type *i* running on a machine of type *j*
- $APC_{\emptyset j}$ − idle power consumption for a machine of type *j*

Objective Functions

- makespan (lower bound)

$$MS_{LB} = \max_j F_j$$

- energy consumed by the bag-of-tasks (lower bound)

$$E_{LB} = \text{execution energy} + \text{idle energy}$$

$$= \sum_i \sum_j x_{ij} APC_{ij} ETC_{ij} + \sum_i \sum_j M_j APC_{\varnothing j} (MS_{LB} - F_j)$$

$$= \sum_i \sum_j x_{ij} ETC_{ij} (APC_{ij} - APC_{\varnothing j}) + \sum_j M_j APC_{\varnothing j} MS_{LB}$$

- note that energy is a function of makespan when we have non-zero idle power consumption

Linear Bi-Objective Optimization Problem

$$\underset{x_{ij},\, MS_{LB}}{\text{minimize}} \begin{pmatrix} E_{LB} \\ MS_{LB} \end{pmatrix}$$

subject to:

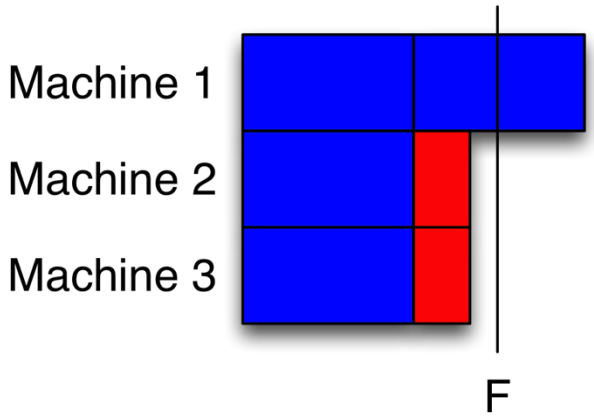
$\forall_i \sum_j x_{ij} = T_i$ task constraint

$\forall_j \; F_j \le MS_{LB}$ makespan constraint

$\forall_{ij} \; x_{ij} \ge 0$ assignments must be non-negative

Linear Programming Lower Bound Generation

- solve for $x_{ij} \in \mathbb{R}$ to obtain a lower bound
  - generally infeasible solution to the actual scheduling problem
  - a lower bound for each objective function (tight in practice)
- solve for $x_{ij} \in \mathbb{Z}$ to obtain a tighter lower bound
  - requires branch and bound (or similar)
    - typically this is computationally prohibitive
  - still making the assumption that tasks are divisible among machines



Recovering a Feasible Allocation: Round Near

- find $\dot{x}_{ij} \in \mathbb{Z}$ that is "near" to x_{ij} while maintaining the task assignment constraint
- for task type (each row in x) do
  - let $n = T_i - \sum_j \text{floor}(x_{ij})$
  - let $f_j = \text{frac}(x_{ij})$ be the fractional part of x_{ij}
  - let set K be the indices of the n largest f_j
  - $\dot{x}_{ij} = \text{ceil}(x_{ij})$ if $j \in K$ else $\text{floor}(x_{ij})$

$x_i =$	3	0	6	5	0
$\dot{x}_i =$	3	0	6	5	0

$x_i =$	3	2.3	6.3	5.4	0
$\dot{x}_i =$	3	2	6	**6**	0

$x_i =$	3	0	6.6	5.4	0
$\dot{x}_i =$	3	0	**7**	5	0

$x_i =$	3.9	2.2	6.4	5.3	4.2
$\dot{x}_i =$	**4**	2	**7**	5	4

Recovering a Feasible Allocation: Local Assignment

- given integer number of tasks for each machine type
- assign tasks to actual machines within a homogeneous machine type via a greedy algorithm
- for each machine type (column in $\dot{x}$) do longest processing time algorithm
  - while any tasks are unassigned
    - assign longest task (irrevocably) to machine that has the earliest finish time
    - update machine finish time

Pareto Front Generation Procedure

- **step 1** weighted sum scalarization
  - **step 1.1** find the utopia (ideal) and nadir (non-ideal) points
  - **step 1.2** sweep α between 0 and 1
    - at each step solve the linear programming problem

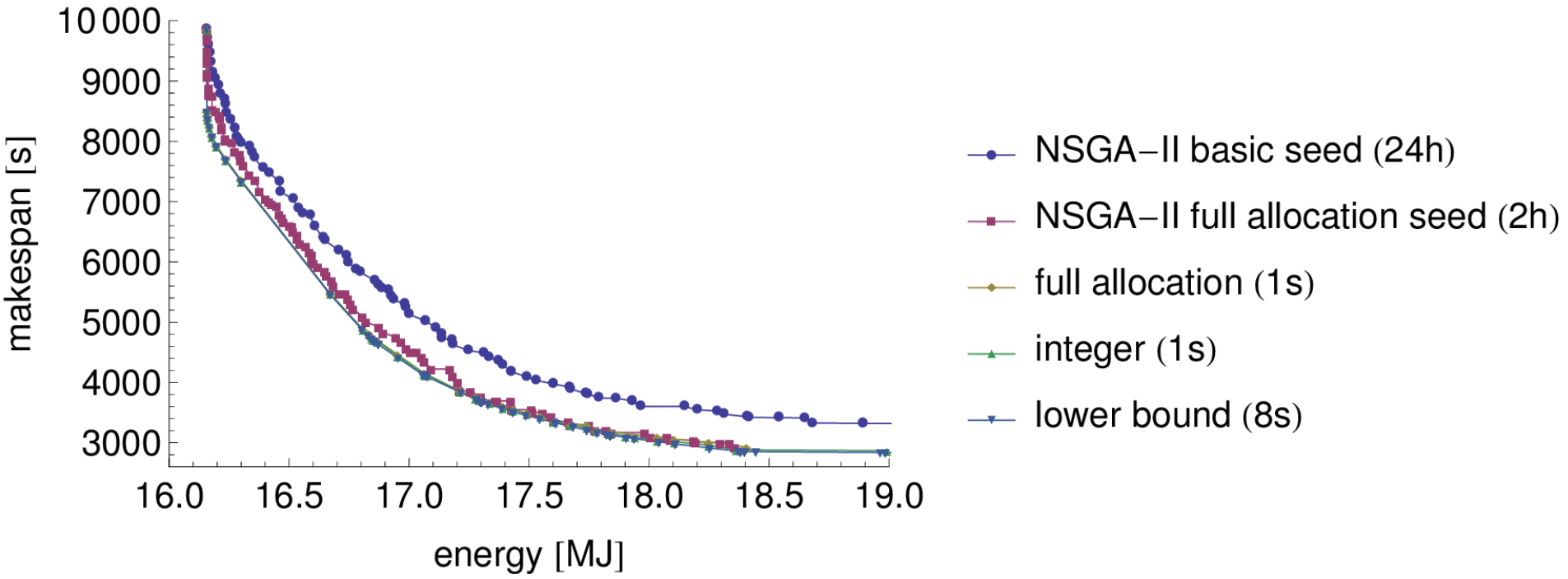
$$\min \frac{\alpha}{D E_{LB}} E_{LB} + \frac{1-\alpha}{D MS_{LB}} MS_{LB}$$

  - **step 1.3** remove duplicates
- linear objective functions and convex constraints
  - convex, lower bounds on Pareto front
- **step 2** round each solution
- **step 3** remove duplicates
- **step 4** locally assign each solution
- **step 5** remove duplicates and dominated solutions
- full allocation is an upper bound on the true Pareto front

Simulation Results

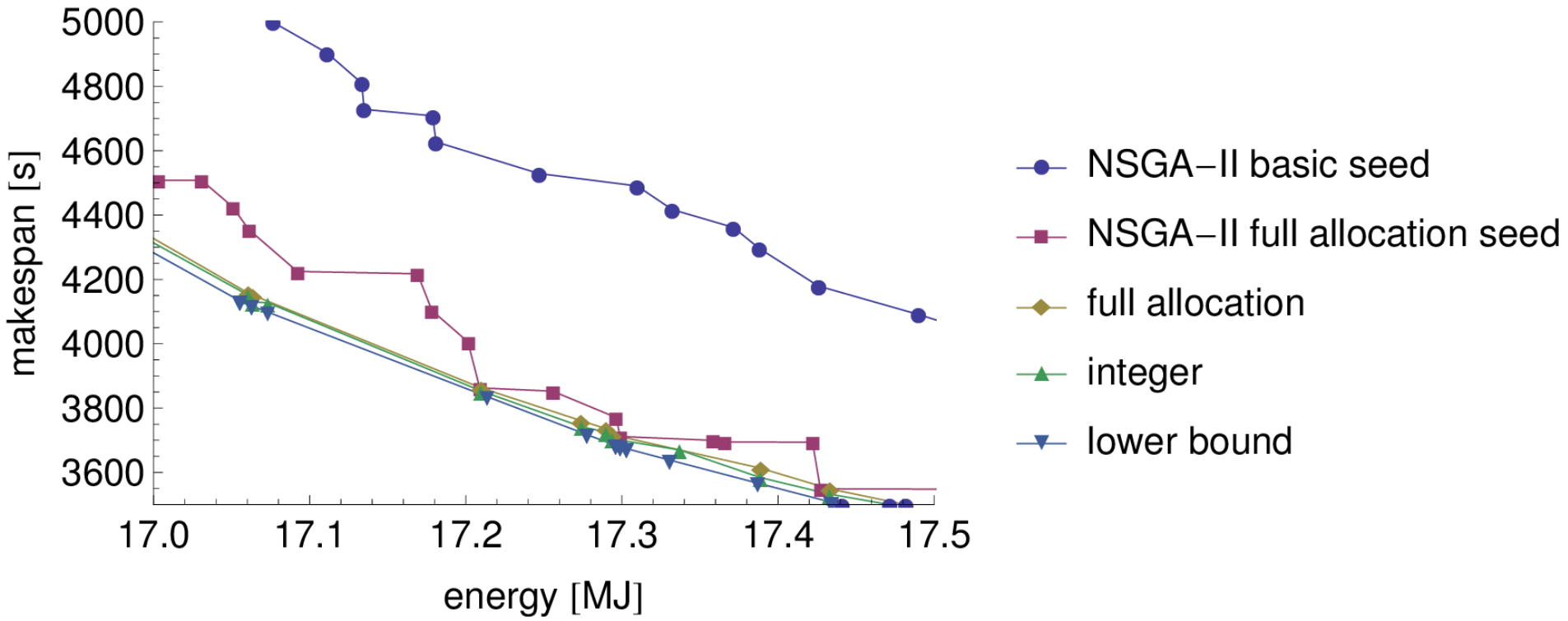
- simulation setup
  - ETC matrix derived from actual systems
  - 9 machine types, 36 machines, 4 machines per type
  - 30 task types, 1100 tasks, 11-75 tasks per type
- compared to NSGA-II
  - basic seed
    - min energy, min-min completion time, and random
  - full allocation seed
    - all solutions from upper bound Pareto front

Pareto Fronts



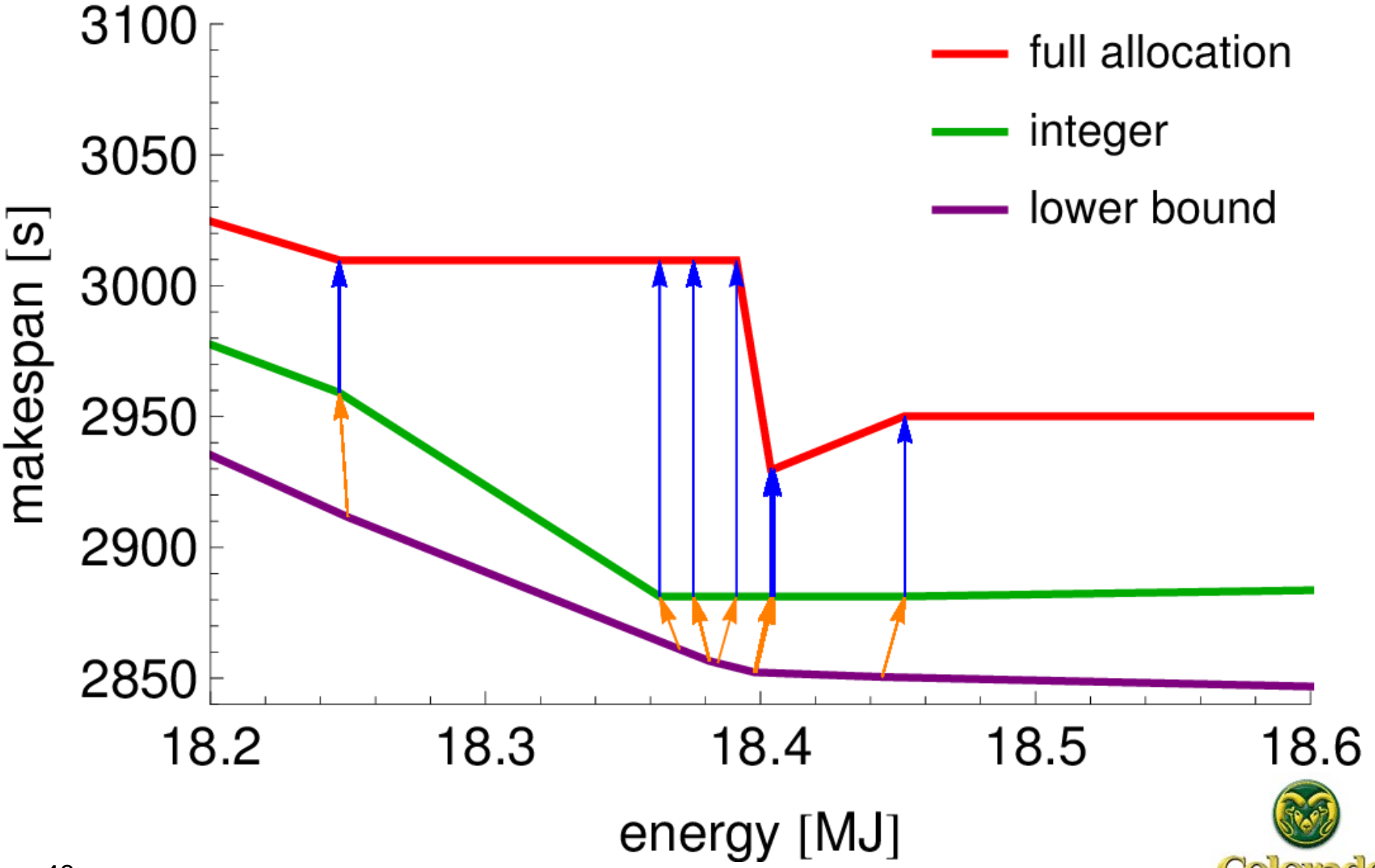
Colorado State University

Pareto Fronts (Zoomed)



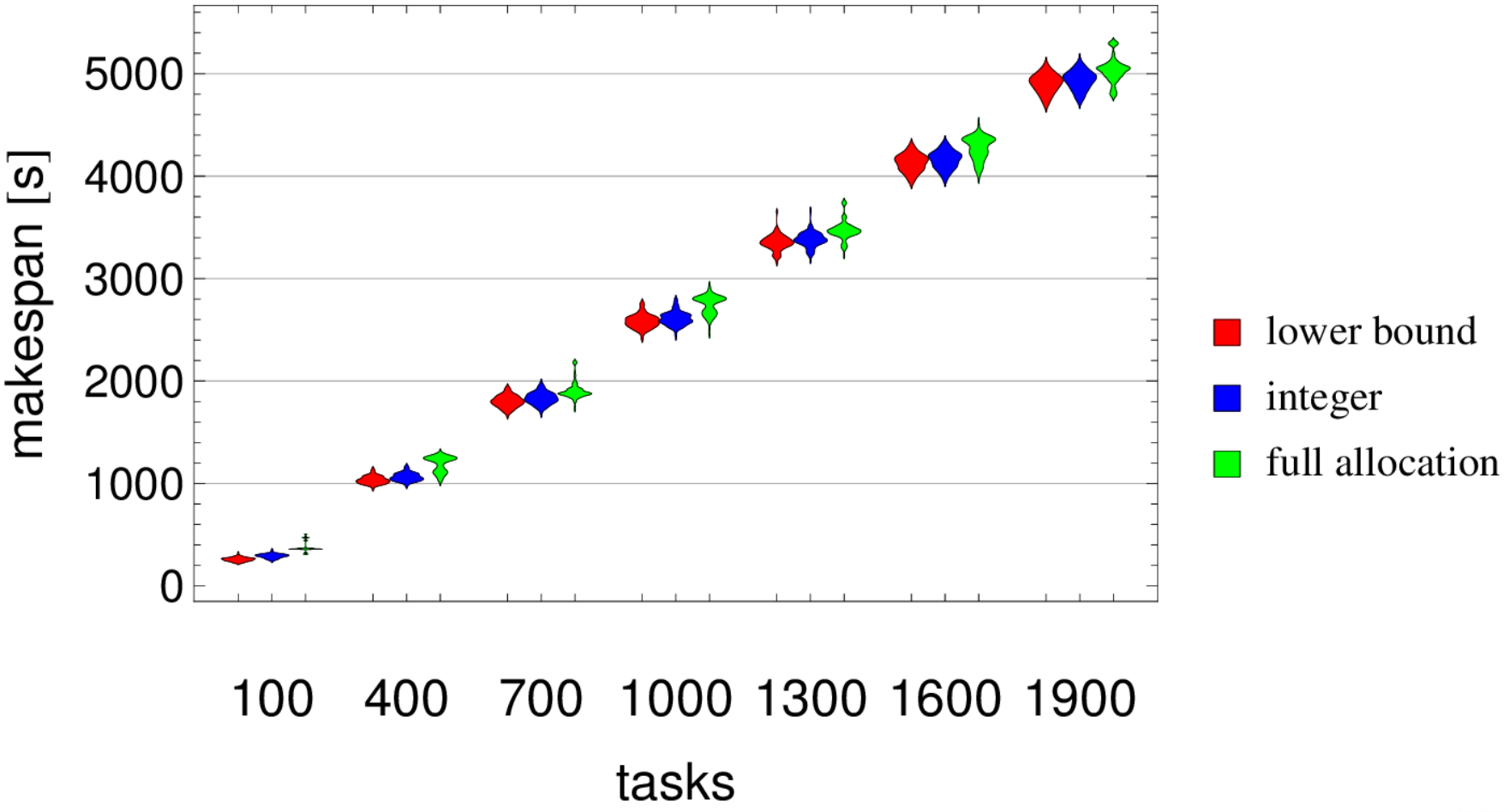
Colorado State University

Lower Bound to Full Allocation, No Idle Power



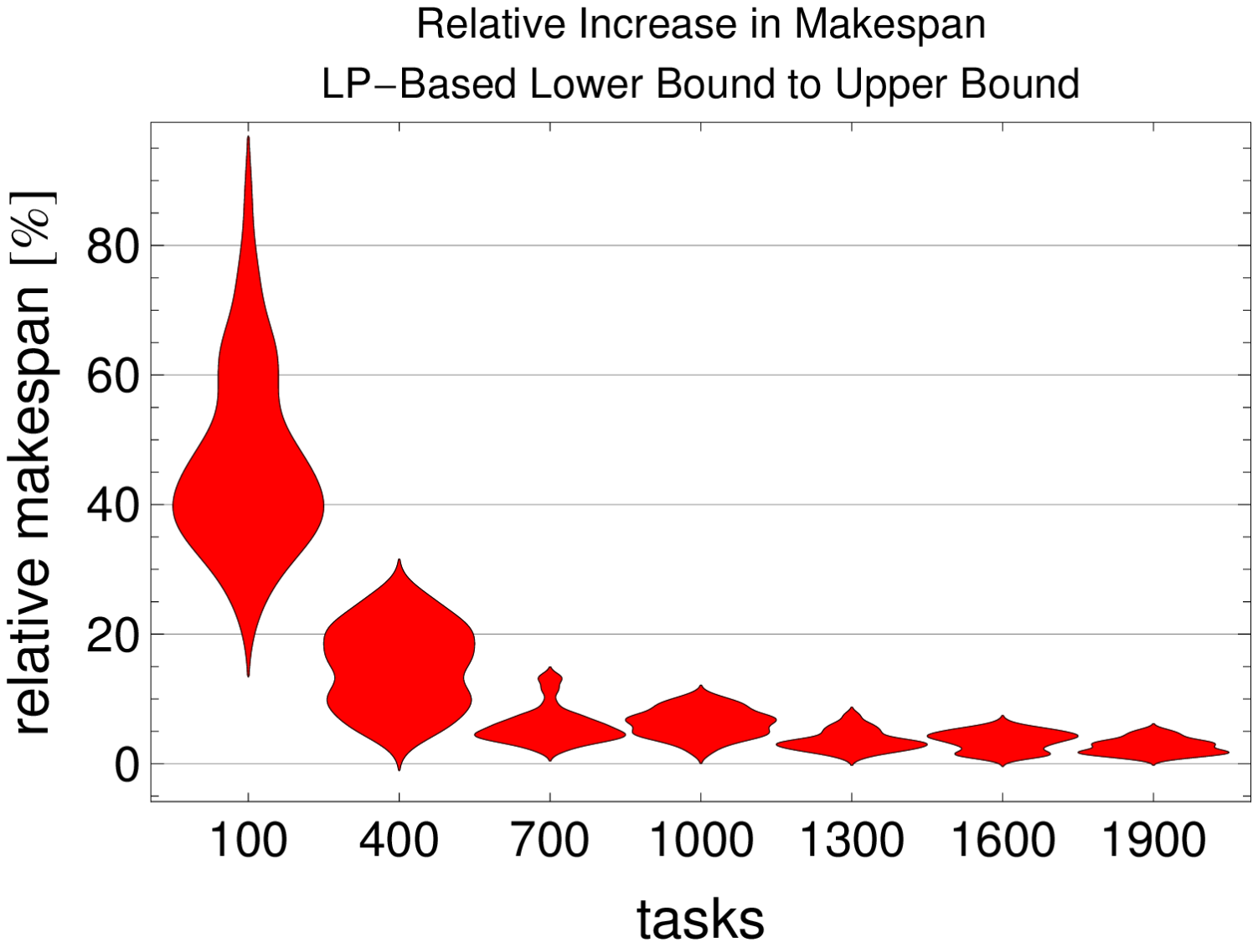
Colorado State University

Linear Programming Based Bounds



Colorado State University

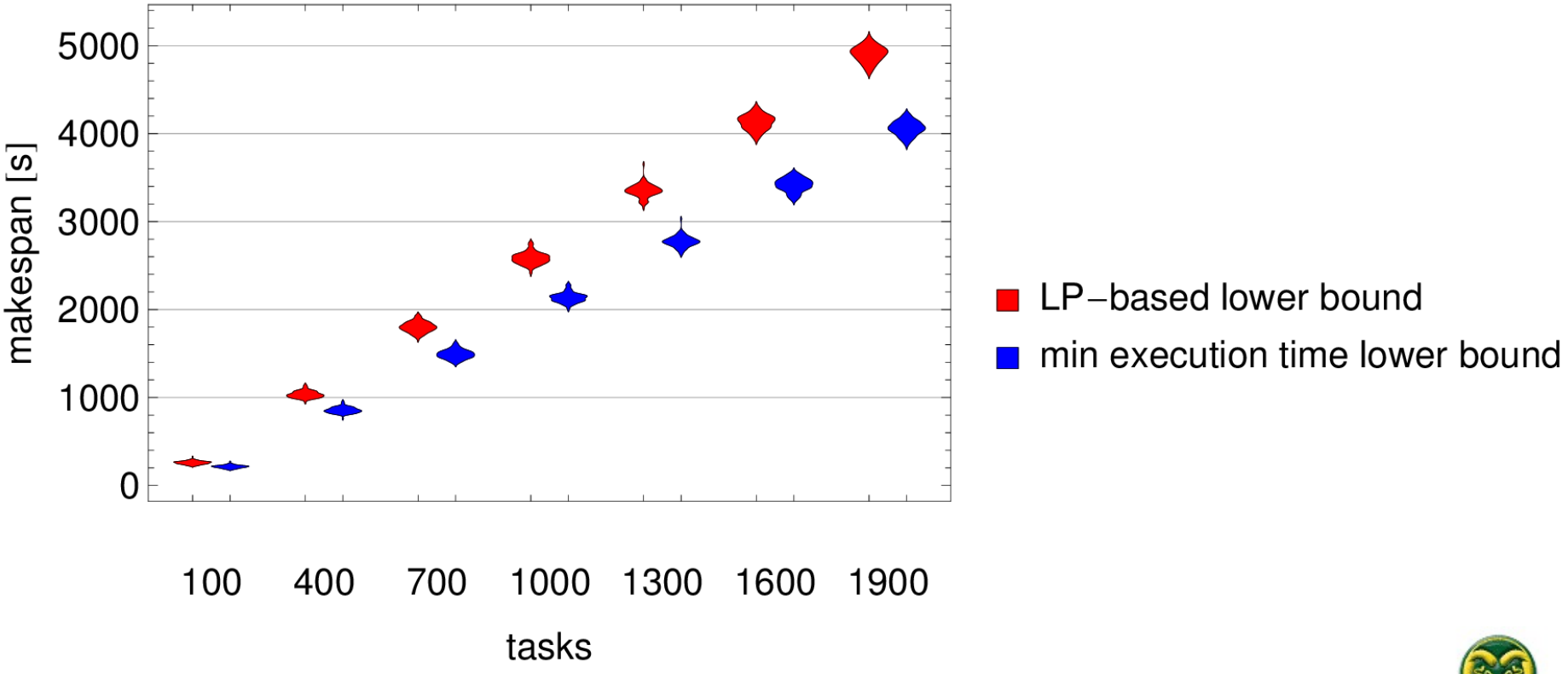
LP-Based Possible Improvement



Colorado State University

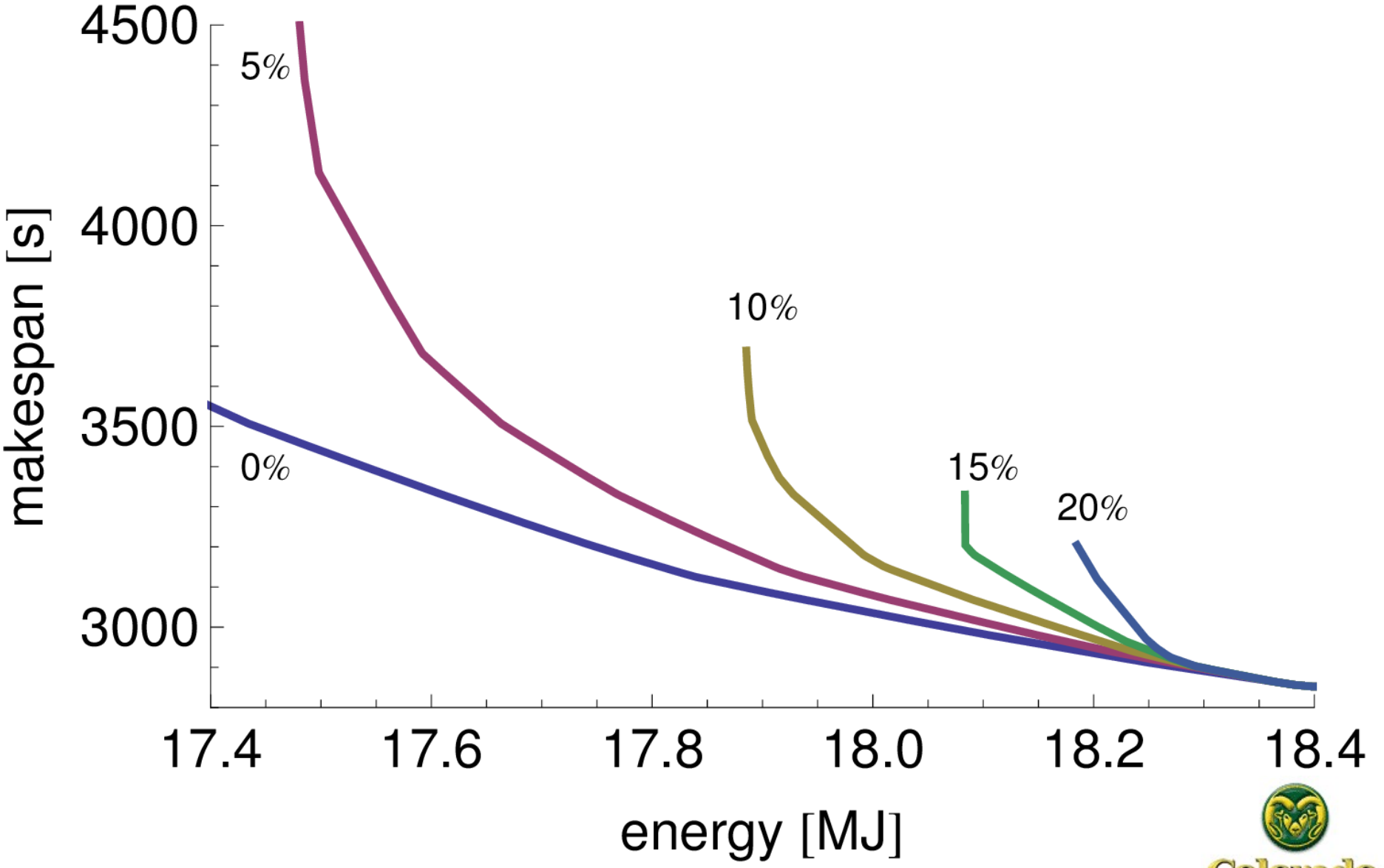
Comparison to Min Execution Time Lower Bound

- assign each task the min execution time machine, divide by number of machines and number of tasks



Colorado State University

Effect of Idle Power

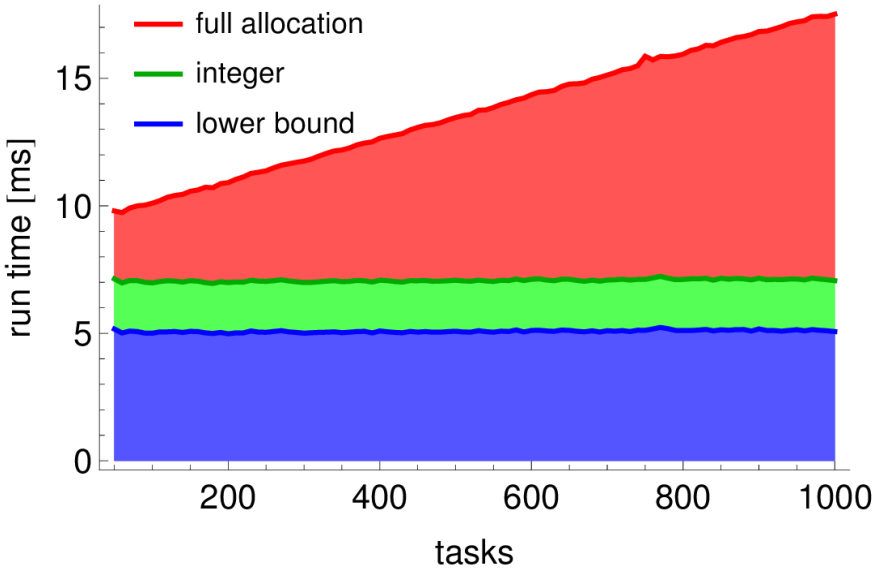


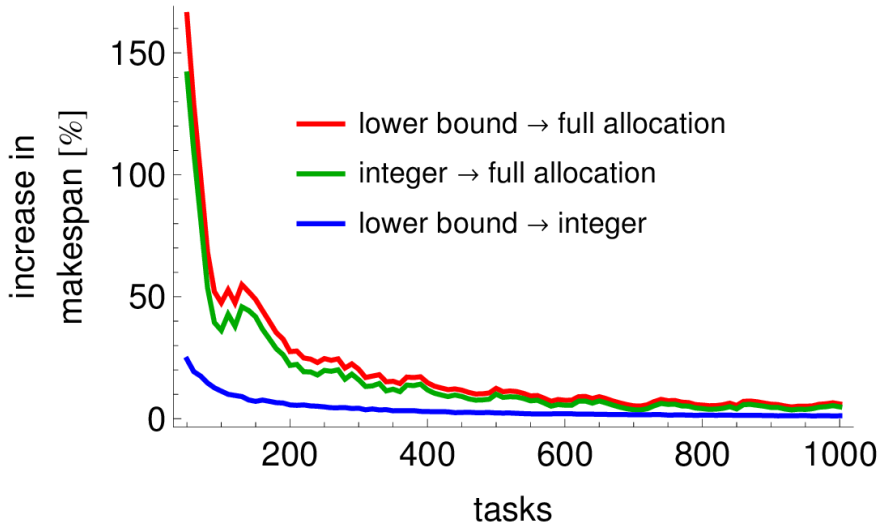
Colorado State University

Complexity

- let the *ETC* matrix be $T \times M$
- linear programming lower bound
  - average complexity $(T+M)^2\ (TM+1)$
- rounding step: $T(M \log M)$
- local assignment step:
  - number of tasks for machine type j is $n_j = \sum_i x_{ij}$
  - worst cast complexity is $M \max_j (n_j \log n_j + n_j \log M_j)$
- complexity of all steps is dominated by either
  - linear programming solver
  - local assignment
- complexity of linear programming solver is **independent** of the number tasks and machines (depends only on the number of task types and machine types)

Impact of Number of Tasks

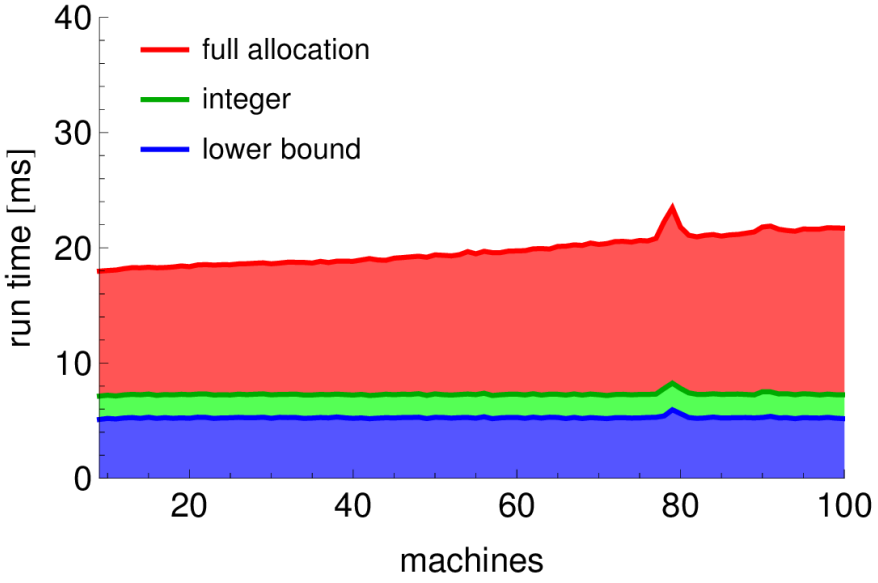


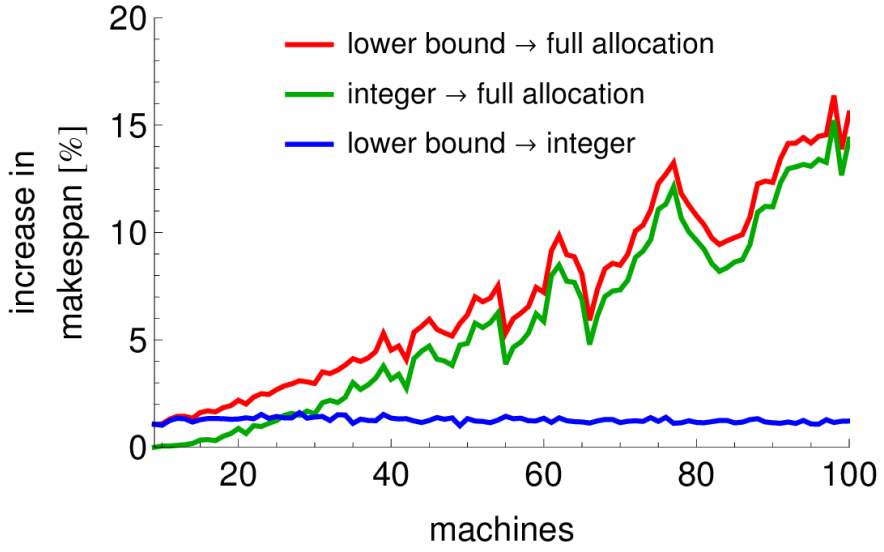


number of task types, machines, and machine types held constant

Colorado State University

Impact of Number of Machines

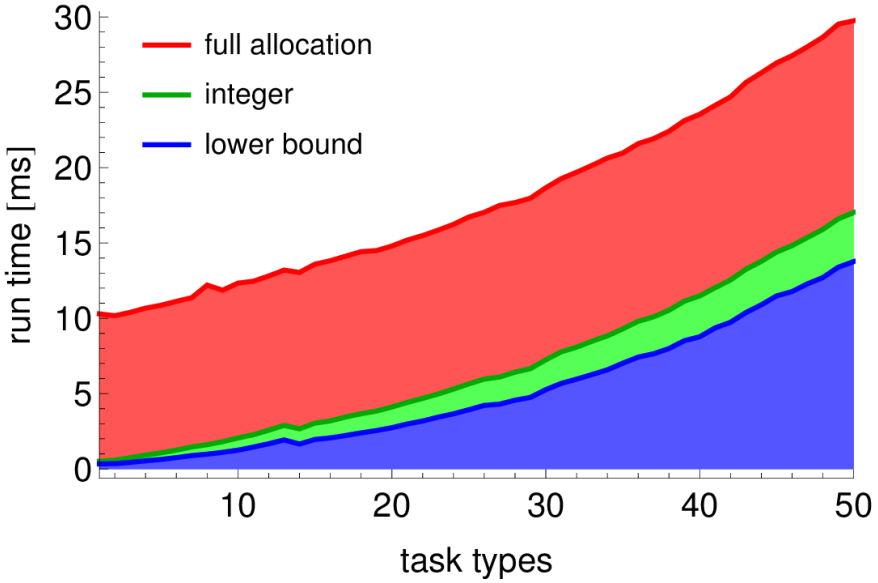


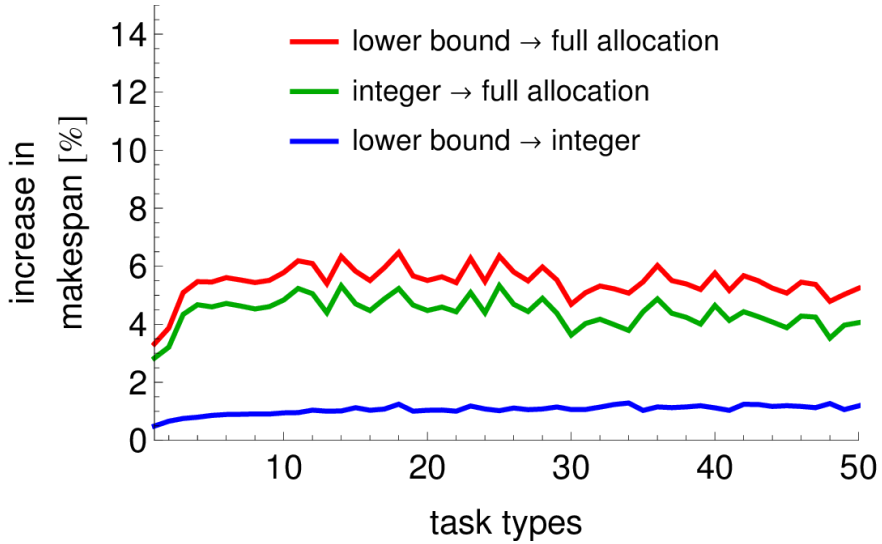


number of tasks, task types, and machine types held constant

Colorado State University

Impact of Number of Task Types





number of tasks, machines, and machine types held constant

Colorado State University

Contributions of Linear Programming Approach

- this linear programming approach to Pareto front generation is efficient, accurate, and practical
  - tight lower bounds on the energy and makespan
  - quickly recovers near optimal feasible solutions
  - high quality bi-objective Pareto fronts

- bounds are tight when
  - a small percentage of tasks are divided
  - a large number of tasks assigned to each machine type and individual machine

- asymptotic solution quality and runtime are very reasonable

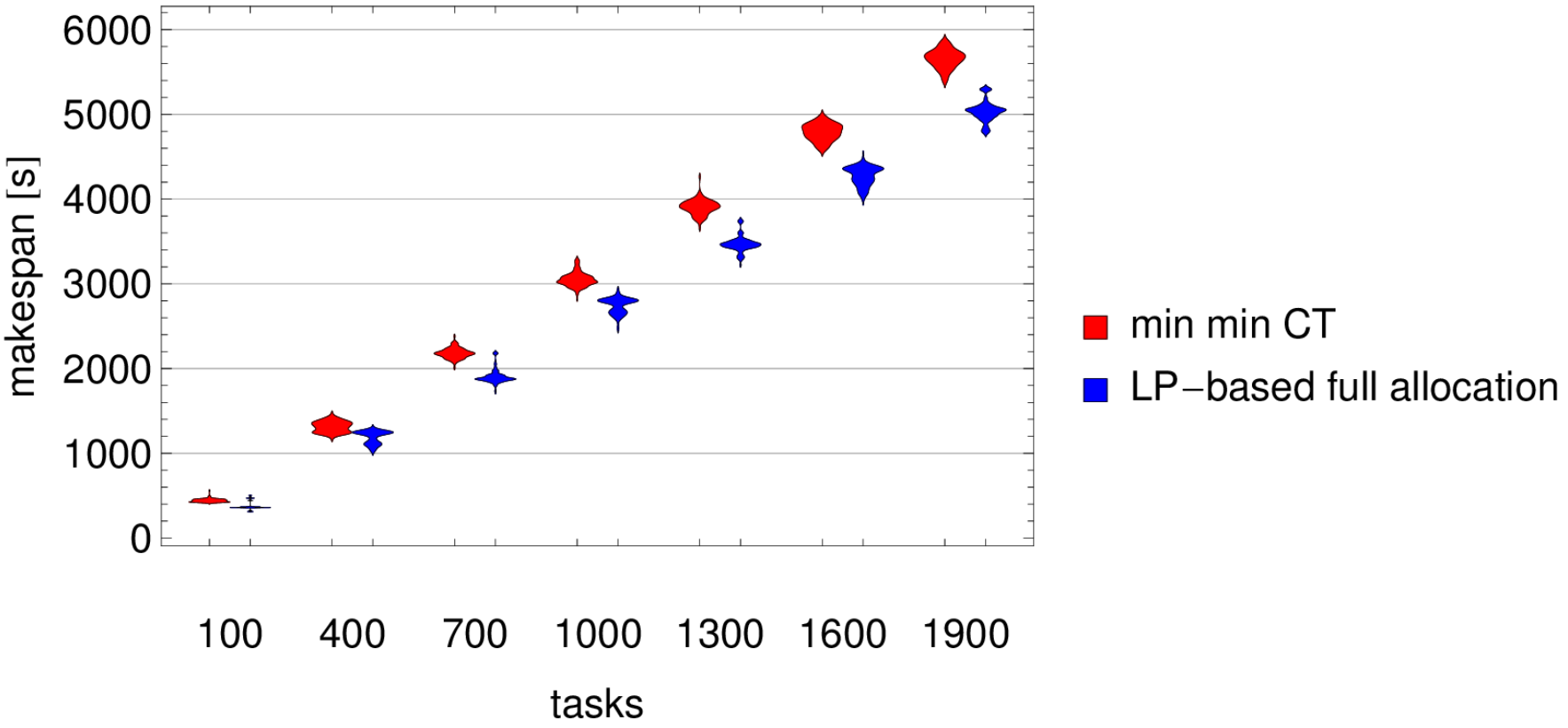
What if you only want one solution on the front?

- How does the linear programming approach compare to a typical greedy scheduling algorithm?

- used COIN-OR linear programming solver (third party library in C++) with lower bound, rounding, and local assignment phases all implemented in C++

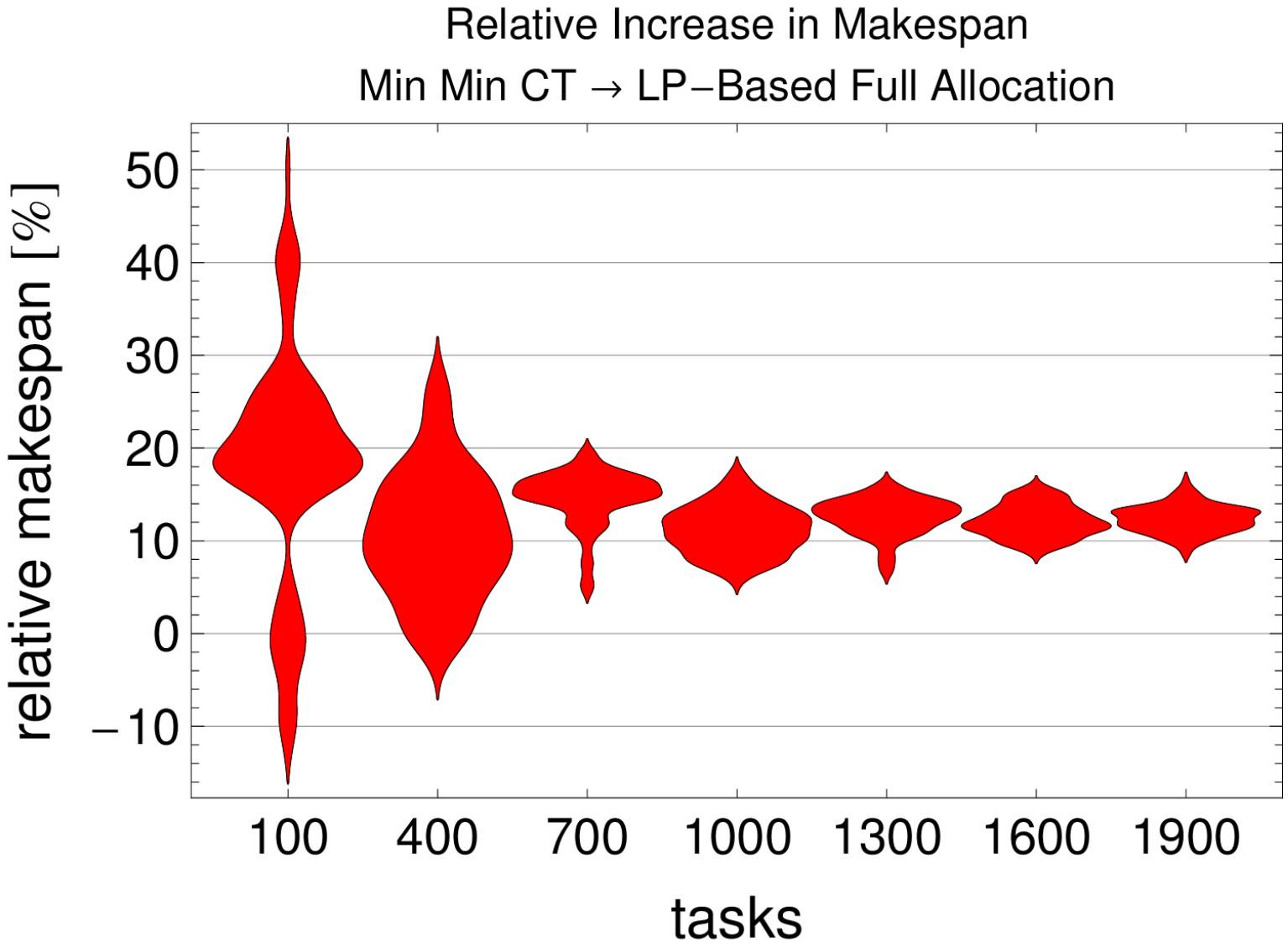
- compare to min-min completion time algorithm in C++
  - finds minimum completion time allocation across all task types and machine types
  - store best machine assignment for each task type, update only those that are assigned last iteration in each pass

Makespan Comparison



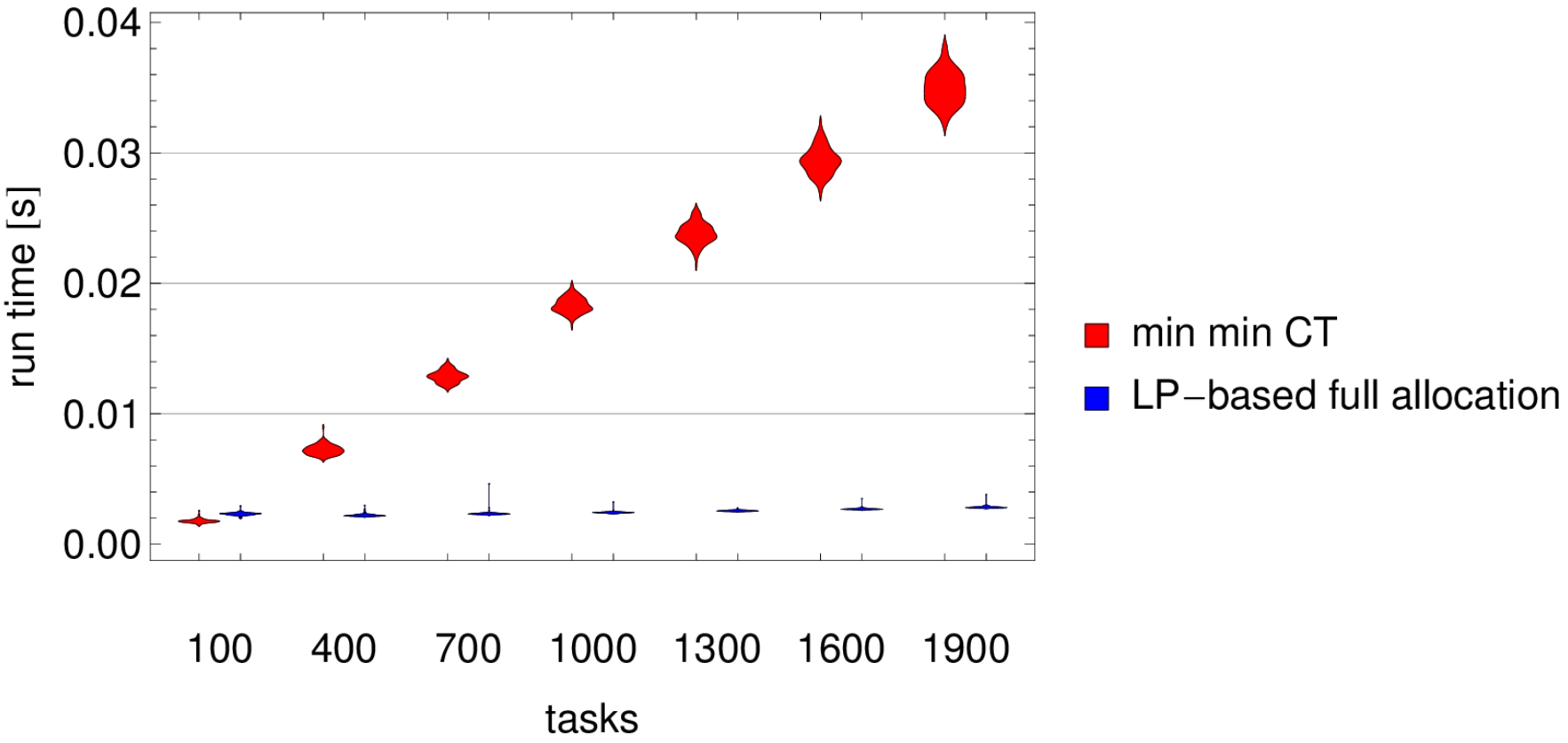
Colorado State University

Makespan Improvement Comparison



Colorado State University

Execution Time Comparison



Colorado State University

Outline

- motivation and environment
- bi-objective optimization
- experimental setup and results
- summary

Summary – Main Points to Remember

- multiple objective problems are very common and extremely important in every day applications
  - today's example: minimize makespan while minimizing energy consumed
    - Pareto fronts allow users to analyze the performance trade-offs between makespan and energy consumed
    - we use Pareto fronts to perform "what-if" analyses to determine the effect of adding or removing machines
- hard problems can not be solved exactly – you have the choice to approximate either the problem or the solution
  - today's example: GA is an approximate solution to the exact problem and LP is an exact solution to an approximated problem

Questions?