

# Scheduling an Industrial Production Facility

Eyjolfur Asgeirsson<sup>1 \*</sup>, Jonathan Berry<sup>2 \*\*</sup>, Cynthia A. Phillips<sup>3 \*\*\*</sup>, David J. Phillips<sup>1 †</sup>, Cliff Stein<sup>1 ‡</sup>, and Joel Wein<sup>4,5 §</sup>

<sup>1</sup> Department of IEOR, Columbia University, New York, NY.

<sup>2</sup> Department of Computer Science, Lafayette College, Easton, PA.

<sup>3</sup> Algorithms and Discrete Mathematics Department, Sandia National Labs, Albuquerque, NM.

<sup>4</sup> Akamai Technologies, 8 Cambridge Center, Cambridge MA 02139

<sup>5</sup> Department of Computer Science, Polytechnic University, Brooklyn, NY.

**Abstract.** Managing an industrial production facility requires carefully allocating limited resources, and gives rise to large, potentially complicated scheduling problems. In this paper we consider a specific instance of such a problem: planning efficient utilization of the facilities and technicians that maintain the United States nuclear stockpile. A detailed study of this problem yields a complicated mixed-integer programming (MIP) model with upward of hundreds of thousands of variables and even more constraints. Consistently and quickly solving such a model exactly is impossible using today's algorithms and computers, and, in addition to branch-and-bound, requires good heuristics and approximation algorithms. In an effort to design such algorithms, we study several different methods of generating good solutions given the solution to the LP relaxation. We design a suite of sample data and test the algorithms. The goals of this project were twofold. First, we wanted to develop a program that could efficiently and accurately help with the Pantex planning problem. Second, we wanted to experimentally test various ideas, designed originally for "cleaner" problems, in this more challenging context. In summary, we demonstrate the value of using  $\alpha$ -points as a way to quickly and cheaply generate, from one solution of an LP relaxation, many feasible solutions to an integer program. In this particular environment, the use of  $\alpha$ -points, combined with other heuristics, outperforms local search. We also see the value of finding combinatorially-structured subproblems as opposed to using simple greedy approaches.

---

\* Research partially supported by NSF Grant DMI-9970063 and an American-Scandinavian Foundation Thor Thors Memorial Fund Grant. [ea367@columbia.edu](mailto:ea367@columbia.edu).

\*\* Some of this work done while this author visited the third author at Sandia National Labs. [berryjw@cs.lafayette.edu](mailto:berryjw@cs.lafayette.edu).

\*\*\* Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000. [caphill@sandia.gov](mailto:caphill@sandia.gov).

† Research partially supported by NSF Grants DGE-0086390 and DMI-9970063. [djp80@columbia.edu](mailto:djp80@columbia.edu).

‡ Research partially supported by NSF Grant DMI-9970063 and an Alfred P. Sloan Foundation Fellowship. [cliff@ieor.columbia.edu](mailto:cliff@ieor.columbia.edu).

§ Research partially supported by NSF Grant DMI-9970063. [wein@mem.poly.edu](mailto:wein@mem.poly.edu).

## 1 Introduction

Managing an industrial production facility requires carefully allocating limited resources, and gives rise to large, potentially complicated scheduling problems. In this paper we consider a specific instance of such a problem: planning efficient utilization of the facilities and technicians that maintain the United States nuclear stockpile. In particular, Sandia National Laboratories has developed and implemented the Pantex Process Model (PPM) [7] to support planning activities at Pantex, a US Department of Energy (DOE) production plant in Amarillo, Texas. The plant simultaneously supports three major DOE programs – nuclear weapon disposal, stockpile evaluation, and stockpile maintenance – which share its facilities, technicians, and equipment. We focus on one piece of the PPM, namely the Evaluation Planning Module (EPM) which projects facility and technician utilization over a given planning horizon (typically a year).

A substantial portion of the Pantex workload relates to evaluation of weapons in the active stockpile. Each weapon evaluation involves partial disassembly of the weapon, one or more inspection tests, and then re-assembly and return of the weapon to the active stockpile.

A detailed study of this problem yields a complicated mixed-integer programming (MIP) model with more than hundreds of thousands of variables and even more constraints. (See Section 2 for a detailed problem description.) Pantex planners want (approximate) answers to planning queries in minutes, with the code running on a local PC. Consistently and quickly solving such a model exactly is impossible using today’s algorithms on a PC, and thus, in addition to using branch-and-bound, we must use good heuristics and approximation algorithms.

In this paper, we empirically evaluate the relative success of several approaches to solving the MIP. We employ traditional branch-and-bound algorithms to exactly solve the MIP. Also, we employ several different techniques in combination to approximately solve this mixed-integer program. These techniques all follow the same general structure:

1. Solve the LP relaxation of the MIP.
2. Use an  $\alpha$ -point method to generate an ordering of the jobs.
3. Convert the ordering into a feasible schedule.
4. Try to improve the schedule via local improvements.

Particularly noteworthy is our use of  $\alpha$ -points in this context (Step 2). Given a solution to a linear program, in which the variables  $x_{jt}$  denote the fraction of job  $j$  completed (or started) at time  $t$ , the  $\alpha$ -point of job  $j$  is the earliest time at which an  $\alpha$  fraction of job  $j$  has been completed (or started). Each value of  $\alpha$  defines an ordering of the jobs; given one LP solution, it is possible to generate many different orderings.  $\alpha$ -points have been used extensively in designing approximation algorithms for combinatorial scheduling problems (see, e.g. [6, 4, 14, 8, 5]), and are explained in more detail in Section 3. Computational work by Savelsbergh, Uma and Wein has shown their practical value in settings where the objective is related to flow time or completion time [13].  $\alpha$ -points have also performed well in practice for the best-effort objective, where one finishes as

many jobs as possible by their deadlines [12]. Möhring, Schulz, Stork, and Uetz use, in addition to other approximation techniques,  $\alpha$ -points to design heuristics for problems with the makespan objective in a setting where jobs also need simultaneous processing [10] from different machines. This feature is the same in the EPM, and is called resource-constrained project scheduling. For a survey of resource-constrained project scheduling, see [1]. To our knowledge, there is no prior work on using  $\alpha$ -points in problems with hard *deadlines*. These problems pose a particular challenge, as a greedy schedule created from a particular job ordering may not satisfy all the deadlines if it must obey precedence and resource constraints. Therefore, the model used by EPM allows (but seeks to minimize) resource overage. Because the EPM plans for future operations, this is not an artificial analysis tool; the goal is to acquire the resources by the time they are needed. Under this objective, we demonstrate the utility of  $\alpha$ -points in a setting with deadlines and precedence constraints.

In a simple combinatorial scheduling problem, once we compute an ordering for the jobs, the task (Step 3) of computing a schedule is typically no more difficult than list-scheduling or some equally simple procedure. In this problem, the ordering is not sufficient to indicate a schedule, so we need to use a more sophisticated job assignment algorithm. We employ two basic approaches for job assignment. One is a greedy method while the other is a more sophisticated approach based on assignment and maximum flow subproblems. By contrasting these two approaches, we are able to evaluate the relative efficacy of using multiple  $\alpha$ -points versus different levels of sophistication of job-assignment. We also employ a local search at the end of the algorithm.

We solve the MIPs within this paper using both the CPLEX 9.0 commercial MIP solver and the parallel Integer and Combinatorial Optimizer (PICO) which is under development at Sandia National Laboratories and RUTCOR [3]. PICO is a general branch-and-bound framework specialized to mixed-integer-programming with many computers in parallel as opposed to the single (or serial) computer version of CPLEX. We describe these issues in more detail in Section 3.

We design a suite of sample data and test the algorithms. We report on our data, experiments and conclusions in Sections 4, 5, and 6.

The goals of this project were twofold. First, we wanted to develop a program that could efficiently and accurately help with the Pantex planning problem. Second, within this challenging context, we wanted to experimentally test various ideas that were originally designed for “cleaner” problems. The details of this paper focus on the second goal, and we report on our particular conclusions in Section 6. In summary, we demonstrate the value of using  $\alpha$ -points as a way to quickly and cheaply generate many feasible solutions to an integer program from one solution of an LP relaxation. In this particular environment, the use of  $\alpha$ -points, combined with other heuristics outperforms local search. We also see the value of finding combinatorially-structured subproblems as opposed to using simple greedy approaches.

## 2 Problem Formulation

In the Pantex planning problem, each job (evaluation) consists of a *set of tasks*, related by precedence constraints. The precedence graph is an *out-forest*. In practice the trees of precedence constraints are highly chainlike. Each task  $j$  has a fixed processing time  $p_j$ , which can be as short as an hour or as long as several months. It also has a release date  $r_j$  and a hard deadline  $d_j$ .

The time horizon is broken into weeks, currently consisting of six consecutive working days. Each day is eight hours long. Task lengths are given in hours.

Each task  $j$  must be performed without interruption in a facility of type  $k_j$ . While task  $j$  is being executed, a technician crew of size  $s_j$  must be assigned to that task. The actual technicians may change over the life of the task, but the crew size must always be maintained. Each technician in the crew must have the proper training to perform the task. Training is controlled by certifications. Each technician has a list of certifications to which (s)he can be assigned and each task  $j$  has a required certification  $c_j$ . Facility availability is given in hours per week and we assume all technicians are available at all times.

In a true schedule, each task is assigned to a specific facility and given a specific team of qualified technicians. However, for planning future technician/facility needs, it is currently sufficient to assign tasks to a pool of facilities and technicians checked at a weekly granularity. That is, a plan is feasible with respect to facility resources for a particular week if the total work for each facility type  $k$  is no more than the number of type- $k$  facility hours available that week. This is equivalent to allowing preemption and migration. Using ideas similar to those used by McNaughton [9], we can show that this underestimates true facility requirements by at most a factor of two, as the preemptive schedule can be converted into a nonpreemptive schedule by taking the one job per machine that is preempted and placing it on a new machine. Technician hours are pooled similarly. To determine certification-hour availability, each technician is assigned to certifications by specifying the amount of time (possibly fractional) (s)he will devote to each certification during each week. No technician is assigned more hours for a particular certification during a week than the total lengths (within that week) of tasks requiring that certification.

We make the further restriction, consistent with Pantex's operations, that short tasks (less than a day) can start at any time of the day, but they cannot be split across two days; they must be completed on the day on which they start.

A *production plan* or schedule assigns a start time to each task. Since preemption is not allowed, a task occupies a facility for its entire duration beginning at its start time. A production plan is feasible if: a) all precedence constraints, release dates, and deadlines are obeyed, b) short tasks are completely scheduled within a single day, c) the total amount of work scheduled for each facility type during any particular week does not exceed the availability of such facilities, d) for each week the requirements for technicians are matched by qualified technician assignments and each technician is assigned no more than a week of work, and e) for each week, no technician is assigned to a particular certification for more hours than the sum of the task lengths (within that week) of tasks requiring

that certification. This last constraint partially prevents serialization of parallel task work, that is, allowing one worker to do the work of two provided he works twice as long.

A typical planning problem spans a year and involves at least 500 jobs and 1000 tasks. Each job has from one to six tasks. There are about 28 – 32 facility types and 300 technicians, each of whom holds 2–5 of the 80 – 100 possible certifications.

In practice, these planning problems are often infeasible. Future work can be performed by newly-hired or retrained technicians in facilities that are newly built or borrowed from other programs sharing Pantex’s facilities. Thus our objective is to find a feasible plan that minimizes the use of extra resources beyond what is currently available. Specifically, we wish to minimize the total number of hours assigned to *ghost facilities (facility overage)* and *ghost certification hours (technician overage)*. In general these terms could be weighted based upon the difficulty of acquiring facilities relative to technicians. However for this study, we weight the two overages equally.

The production plan is not a true schedule because we do not assign workers to specific tasks. We only guarantee that the pool of workers is sufficient in the aggregate. Given the uncertainty in the data (job load, task duration, etc), extra resolution is probably not warranted. However, the optimal overage for this schedule is a lower bound on the optimal overage for a true schedule where tasks are assigned to specific facilities with specific teams of technicians and all resources constraints are met at the finest resolution (in this case, by the hour). Therefore, a decision to allocate more resources to Pantex can be justified based on the optimal plan, but the resources specified in the plan may not be sufficient.

## 2.1 MIP Formulation of the Pantex Problem

### Input parameters/constants/shorthand

$p_j, r_j, d_j$	The processing time (in hours), release day and deadline day of task $j$
$\rho_j$	The minimum number of full days needed to process task $j$ (i.e. $\rho_j = \lceil p_j/8 \rceil$ ).
$T(j)$	Set of possible start days for task $j$ .
$p(j, t, \tau)$	Number of hours work on task $j$ performed during week $\tau$ if $j$ is started on day $t$ .
$T(j, \tau)$	Set of start times $t$ for task $j$ such that $p(j, t, \tau) > 0$ .
$k_j, c_j, s_j$	Facility type, technician certification and crew size for task $j$ .
$C(w)$	The set of certifications held by worker $w$
$f_{k,\tau}$	The number of hours available in facilities of type $k$ during week $\tau$ .
$b(j, j')$	1 if $j \prec j'$ and task $j'$ can be “packed” with $j$ , 0 otherwise

### Variables

We use four types of variables. The  $x$  variables are integral, while the remaining variables are rational.

$x_{jt}$	= 1 if task $j$ starts on day $t$ , and 0 otherwise
$y_{wc\tau}$	= fraction of time worker $w$ spends using certification $c$ in week $\tau$
$F_{k\tau}$	= number of ghost facilities of type $k$ in week $\tau$
$G_{c\tau}$	= number of ghost technicians with certification $c$ in week $\tau$

### MIP formulation

$$\text{(TILP) minimize } \sum_{k,\tau} F_{k\tau} + \sum_{c,\tau} G_{c,\tau}$$

subject to

$$\sum_{t \in T(j)} x_{jt} = 1 \quad \forall j \quad (1)$$

$$\sum_{c \in C(w)} y_{wc\tau} \leq 1 \quad \forall w, \tau \quad (2)$$

$$x_{jt} \leq \sum_{r_{j'} \leq t' \leq t - \rho_{j'} + b(j', j)} x_{j't'} \quad \forall t \in T(j), j' \prec j \quad (3)$$

$$\sum_{j:k=k_j} \sum_{t \in T(j,\tau)} p(j, t, \tau) x_{jt} \leq f_{k,\tau} + 48F_{k\tau} \quad \forall \tau, k \quad (4)$$

$$\sum_{j:c_j=c} \sum_{t \in T(j,\tau)} p(j, t, \tau) s_j x_{jt} \leq \sum_w 48y_{wc\tau} + 48G_{c\tau} \quad \forall \tau, c \quad (5)$$

$$48y_{wc\tau} \leq \sum_{j:c_j=c} \sum_{t \in T(j,\tau)} p(j, t, \tau) x_{jt} \quad \forall c, \tau, w : c \in C(w) \quad (6)$$

$$x \in \{0, 1\} \quad (7)$$

Constraints (1) ensure that every task is done. Constraints (2) prevent over-scheduling any technician in any week. Constraints (3) ensure a task is not started until all predecessors are completed. Constraints (4) ensure there is sufficient facility capacity in each week to perform all the work that must be done in that week, and constraints (5) are the analogous constraints on certification hours. Constraints (6) prevent some situations where a technician is taking the place of multiple technicians. There are 48 work hours in a week. Although short jobs may start at any hour, as long as they fit in a day, we will always start them at the beginning of the day or immediately following the completion of their last predecessor (whichever is later). We will refer to this program as TILP.

### 3 Algorithms

In this section, we describe our algorithms. We also describe the branch and bound used to solve the mixed integer program TILP, which provided the optimal objective values (or lower bounds) that we use as a basis for comparison for our heuristics. Our heuristics all followed the same general steps:

1. Solve the linear programming relaxation of TILP to obtain  $x_{jt}^*$ .
2. Call the BEST- $\alpha$  or FIXED- $\alpha$  subroutine to convert the TILP solution  $x_{jt}^*$  to a set,  $\Pi$ , of priority orderings for the jobs
3. For each ordering  $\pi \in \Pi$ , call the COMB or GREEDY subroutine to determine a schedule  $\omega(\pi)$  for the tasks
4. Use local search to try to improve the objective for the schedules.

Steps 2 and 3 each employ two different subroutines resulting in four different algorithms. We implemented and tested each algorithm, and also examined the value of using local search to improve the solution.

*Mixed-integer programming* To reduce the size of the MIP, task starts are represented at daily granularity rather than to the hour (the smallest size of a task). We force long jobs to start at the beginning of a day. This does not significantly decrease planning flexibility. However, if we were to force short jobs to align with the start of days, we could introduce significant unforced idle time, especially on chains of short jobs. Therefore, short jobs can start at hourly increments within a day. This is not explicitly represented in the MIP. Start times are interpreted as mid-day (earliest possible) whenever one or more predecessors are also running that day.

To enforce this interpretation, we must take some care in formulating precedence constraints. In the MIP given in Section 2, we enforce precedence  $j \prec j'$  in constraints (3). Normally, we need only enforce a precedence constraint between immediate (predecessor,successor) pairs. However, when  $j$  and  $j'$  can finish on the same day (they pack,  $b(j, j') = 1$ ), we must usually add an extra precedence constraint between  $j'$  and a more distant predecessor to guarantee  $j'$  is always scheduled completely within a day.

We used two different MIP solvers, CPLEX 9.0 and PICO, to solve the mixed-integer programs. CPLEX (serial) is the standard commercial MIP solver, whereas PICO (Parallel Integer Combinatorial Optimizer) is a massively-parallel mixed-integer programming solver being developed at Sandia National Laboratories and RUTCOR. PICO requires a linear-programming solver to compute lower bounds, for which we currently use the free solver, COIN LP. PICO is licensed under the GNU general library public license and is expected to be released free in 2004. We used only the serial version of PICO in this paper.

Neither solver could handle problems spanning the entire year-long time horizon, and so we tested our algorithms on problems spanning roughly half the time horizon (i.e. around six months). For these problems, both codes could solve most, but not all of the instances considered, and CPLEX was faster than PICO. Sections 4 and 5 contain more details about the size of problem versus tractability.

We also started developing routines specific to the MIP for use within PICO. For a fractionally assigned task  $j$ , we can branch on the full set of  $x_{jt}$  variables where one child forces  $j$  to start by time  $t^*$  and the other child forces  $j$  to start after  $t^* + 1$  where  $t^*$  is the  $\alpha$ -point for  $\alpha = 0.5$ . One could get similar behavior with CPLEX by explicitly listing all decision variables for each job as a special-ordered set. We make branch choices based on gradients just as it would for branching on single variables. As described in Section 3, we can use an  $\alpha$ -point-based algorithm to find feasible solutions early. This allows early pruning of the search tree.

In future work, we will use the parallel version of PICO in order to solve the full version of these problems, and test the ideas that we have implemented within PICO.

*$\alpha$ -points* The IP code described in the previous section is not able to efficiently solve all of our instances. We therefore begin with the solution  $x_{jt}^*$  to the linear programming relaxation. In step 2, BEST- $\alpha$  and FIXED- $\alpha$  convert a solution  $x_{jt}^*$  to a set of orderings,  $\Pi$ , of the tasks. They both use the method of  $\alpha$ -points to do this. The variables  $x_{jt}$  were defined in the formulation of TILP, and are an indicator of which day a given task starts in. After relaxing the integrality, the start-time variables  $x_{jt}$  now represent the fraction of the task  $j$  that starts at day  $t$ . Start-time variables can be converted into the more standard completion-time variables by the formula:  $\bar{x}_{j(t+\rho_j)} = x_{jt}, \forall j, t$ . Then, for a given  $\alpha \in [0, 1]$ , and solution to the linear programming relaxation of (TILP), the  $\alpha$ -point of a task,  $\bar{t}_j^\alpha$ , is defined as the first day  $t$  at which the fraction of task  $j$  completing at or before  $t$  exceeds  $\alpha$ . Formally, this is:

$$\bar{t}_j^\alpha = \min\{t : \sum_{\tau \leq t} \bar{x}_{j\tau} \geq \alpha\}. \quad (8)$$

Alternatively, we can use the original time-indexed variables in equation (8):

$$t_j^\alpha = \min\{t : \sum_{\tau \leq t} x_{j\tau} \geq \alpha\}. \quad (9)$$

Thus,  $t_j^\alpha$  represents the first day  $t$  at which the fraction of task  $j$  starting at or before  $t$  exceeds  $\alpha$ . The task ordering is simply the (ascending) order of either  $t_j^\alpha$  or  $\bar{t}_j^\alpha$ . In general, using the orderings generated by  $\bar{t}_j^\alpha$  will favor starting longer tasks later than shorter tasks. Our experiments showed that  $t_j^\alpha$  resulted in a better objective than  $\bar{t}_j^\alpha$ . This may be a result of the presence of due dates and the resource optimizing objective, which would favor starting longer tasks earlier.

Implementing these methods involves calculating  $t_j^\alpha$  or  $\bar{t}_j^\alpha$  from equations (9) or (8) for each  $\alpha$  in some given set  $\mathcal{A}$ . The distinction between different  $\alpha$ -points methods involve what  $\mathcal{A}$  is. Our heuristics used methods known as BEST- $\alpha$  and FIXED- $\alpha$  (see e.g. [13]). BEST- $\alpha$  corresponds to trying all  $\alpha \in [0, 1]$  that yield different orderings. There are a finite number of  $\alpha$  to try, as the orderings change at most once for every nonzero  $x_{jt}$ . For a basic solution to MILP, the number of nonzero  $x_{jt}$  is bound by the number of basic variables in the linear programming relaxation. For our particular implementations, we approximated BEST- $\alpha$  by using 51  $\alpha$  spread uniformly over the  $[0, 1]$  interval. The FIXED- $\alpha$  method corresponds to setting  $\mathcal{A}$  to can just one particular  $\alpha$ . We set  $\mathcal{A} = \{0.5\}$ .

**COMB** This section describes an algorithm, COMB, that converts an ordering,  $\pi$ , of the tasks to a feasible schedule, or plan,  $\omega(\pi)$ . In addition to the ordering, COMB uses information from the linear program relaxation and combinatorial structure to create schedules. This algorithm is the one used to generate early feasible solutions to strengthen pruning in (PICO-based) branch and bound.

Given an ordering  $\pi$ , the heuristic must first deal with a subtlety concerning the discretization of the problem. Specifically, for a short task  $j'$  that packs with its predecessor  $j$ , the starting time alpha point  $t_{j'}^\alpha$  (in this case equivalent to  $\bar{t}_{j'}^\alpha$ ) might be the same day in which task  $j$  finishes. Since the alpha points are

expressed in days, not hours, task  $j$  completes sometime in the middle of day  $t_j^\alpha$  when started at the beginning of day  $t_j^\alpha$ . The true starting hour of task  $j'$  must be adjusted accordingly.

COMB computes the “resource availability” from the linear programming solution. Ghost facilities used in the LP solution are added to the true facility availability. Technician-hour availability for each certification for each week is the number of hours the LP assigns to that certification over all technicians for that week plus the ghost technician hours. Since the objective-function value of this solution is a lower bound on the optimal integer solution, we are free to use all these resources in our heuristic solution without degrading the objective value. Therefore, only resources used in excess of this availability are considered “overage” for this heuristic.

COMB schedules tasks in alpha-point order. To schedule a task  $j$ , COMB computes the incremental decrease in technician and facility availability incurred by each potential starting day of  $j$  and places  $j$  in the earliest place that creates no new overage. If no such place exists, COMB places the task either as early as possible, or in the place with minimum overage. The strategy is determined at the start of the heuristic; for this paper we try both and take the best result. COMB decrements the technician and facility availability for each week in the lifetime of the task and continues to the next task.

Given the greedily-determined start times, COMB computes an optimal (rational) technician assignment using one maximum-flow computation for each week. The flow problem is formulated as follows. There is a source node  $s$ , a node  $W_i$  for each worker  $i$ , a node  $C_j$  for each certification  $j$  and a sink  $t$ . The source is connected to each of the  $W_i$  nodes with capacity equal to the total time the worker is available to work during this week. In our current model, this is the same for all workers: 48 hours. There is an edge from each worker node  $W_i$  to each certification node  $C_j$  where worker  $i$  has certification  $j$ . The capacity is the total number of hours of work for certification  $j$  scheduled in this week (according to our heuristic schedule). Finally, there is an edge from each certification node to the sink with capacity equal to the total man-hours of work for certification  $j$  in this week. That is, for each (piece of a) task with certification  $j$  run in this time week, we multiply the length of the task (in this week) by the crew size. The capacity on the source-to-worker edges reflects the bound on technician availability. The capacity of the worker-to-certification edges reflects the total time a worker can spend on a certification (constraints 6 in MILP). The capacity of the certification-to-sink edges reflects the total work requirement for each certification. The technician assignment is encoded in the flow from worker nodes to certification nodes. The residual capacity on the edge from certification  $j$  to the sink (that is the difference between the capacity and the flow) is the technician overage from this assignment. In particular, if the maximum flow in this network saturates all the certification-to-sink edges, then all the work is done and there is no overage.

*Assigning resources greedily* The greedy subroutine for assigning resources, GREEDY, creates a feasible schedule,  $\omega(\pi)$ , from a given ordering,  $\pi$ . Note that  $\omega(\pi)$  corre-

sponds to a start time for each task, a schedule for each technicians and facility, and a number of ghost hours needed for each certification and facility per week.

We either assign all technicians to tasks first and then all facilities to tasks, or vice versa. Our experiments demonstrated the former is better. Because of these results, and the two methods' similarity, we only describe the TechSchedFirst subroutine. Due to space constraints we do not prove the algorithm's correctness.

Throughout TechSchedFirst, a set of jobs,  $\mathcal{J}$ , which are not scheduled and have no uncompleted predecessors are maintained. Also, each job's earliest possible start time is maintained. This is the minimum of its release date and the completion time of its predecessors. At each iteration, TechSchedFirst greedily assigns technicians to the job in  $\mathcal{J}$  with the highest priority according to  $\pi$  by determining the time in the time window of its earliest possible start time and due date that corresponds to the point at which the most technicians will be free. This is possible to do via interval trees (see Cormen et. al. 2001, [2]). If more technicians are required, the necessary amount of ghost technician hours are allocated. TechSchedFirst then updates  $\mathcal{J}$  and the earliest possible start times of all jobs. With the start times of each job established, TechSchedFirst then schedules facilities to each task in ascending start time order, with ghost facility hours scheduled as needed.

*Local search* Since our experiments indicated that technicians were a more constrained resource, we implemented a straightforward local search heuristic, called LOCALIMPROVE, that swaps technicians between different tasks. We did not implement local search on COMB as our goal was to compare the more sophisticated job-assignment heuristic to greedy methods.

LOCALIMPROVE takes a feasible schedule as input and returns a feasible schedule with an objective no worse than the input. LOCALIMPROVE assumes tasks are sorted in descending ghost technician hours. The subroutine examines each task  $j$ , and, if it has ghost technicians, determines any other tasks that have scheduled process time overlap. It calculates the set,  $\mathcal{C}^C$ , of all tasks that have no overlap with  $j$ . Then  $\mathcal{C} = \mathcal{J} \setminus \mathcal{C}^C$  is the set of tasks with overlap with  $j$ . Swapping technicians to task  $j$  only affects tasks in  $\mathcal{C}$ . It finds all technicians that are not currently working on  $j$ , but have the correct certification to work on  $j$ . It swaps the technician to  $j$  that corresponds to the best change in the objective, updating the ghost requirements of both  $j$  and any task in  $\mathcal{C}$  on which the technician was working. This is iterated as long as  $j$  still has ghost technicians.

We also applied a simple evolutionary algorithm which did not work as well. In future work, we will consider more sophisticated evolutionary algorithms.

## 4 Data

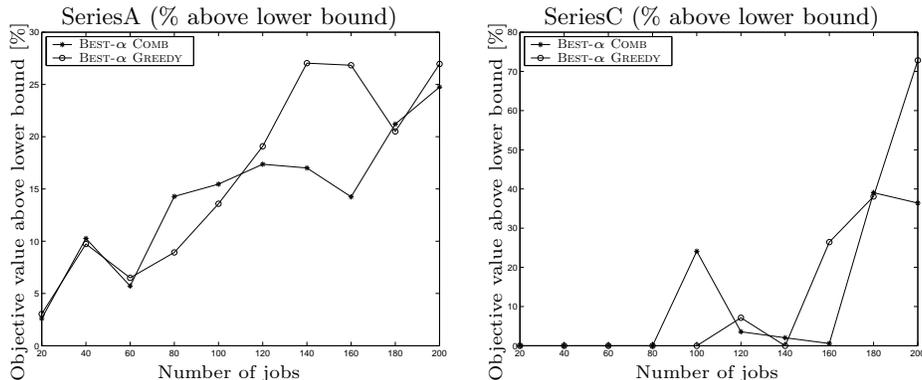
Due to security reasons, access to actual data was severely restricted. However, we know many properties and characteristics of the real datasets. We created a problem generator that is easily modified and creates problem instances that capture properties of the real data.

Series (jobs)	Tasks	Crew	Certs	Techs	C.per.T.	Fac.Types	Fac.Hours
SeriesA - (20)	212	1-6	100	300	3-5	30	100
SeriesA - (100)	1060	1-6	100	300	3-5	30	100
SeriesA - (200)	2120	1-6	100	300	3-5	30	100
SeriesB - (20)	212	1-6	100	150	3-5	6	10
SeriesB - (100)	1060	1-6	100	750	3-5	6	50
SeriesB - (200)	2120	1-6	100	1500	3-5	6	100
SeriesC - (20)	212	1-2	100	300	3-5	30	100
SeriesC - (100)	1060	1-2	100	300	3-5	30	100
SeriesC - (200)	2120	1-2	100	300	3-5	30	100

**Table 1.** Overview of selected problems

The problem generator uses predefined job templates that describe the successor-predecessor relationships of the tasks and include parameters for the size and requirements of each task. These templates were created using our knowledge that in the actual problem the jobs are highly chainlike, each task can have at most one predecessor and most tasks have only one successor even though we allow tasks to have more than one successor. We then combined this with templates for the facilities, technicians and certifications and the scenario, which defines the random distributions of the task sizes along with the time horizon. The actual size of the tasks is determined using a uniform random distribution, whose parameters are dependent on the scenario and the task. To select the certification and facility for a task, we split the certifications and facilities into a few sets and then randomly pick a certification and facility.

Due to hardware and time issues we created series of scaled down problems that retain most of the characteristics of the real problems. We needed to make the problems small enough so that the branch and bound could solve most of the instances, while keeping them large enough to contain all the original structure. The time horizon was cut down from a year to 6 months, and job lengths scaled down so that we fit the same number of jobs in this time horizon. To make the problems more challenging, we made the job templates larger and less chainlike than in the actual problem. We created three sets of problems, with 10 problems in each set. The goal of the first dataset, “Series A,” was to see how the algorithms would handle increasingly difficult problems that share the same fundamental properties. In some sense, the problem with 100 jobs is the most like the actual EPM problem. The next set of problems, “SeriesB”, was created to see how the algorithms would deal with problems of increasing size but not increasing difficulty. Since Series A and B seemed to require more technicians than facilities, we created “Series C” by decreasing the number of ghost technicians that the solutions requires and hence increased the importance of the ghost facilities. Table 1 shows a small, medium and large problem of each problem set. Finally, we also generated a full-year problem to test whether our heuristics would be able to solve a problem of this size. To create the full-year problem, we used the Series A problem with 100 jobs and doubled the time



**Fig. 1.** Performance on SeriesA and SeriesC. We show the performance of the algorithms for SeriesA and SeriesC as a percentage of the objective value above the best known lower bound. The best known lower bound is the optimal IP solution for all problems except 160 – 200 jobs of SeriesA.

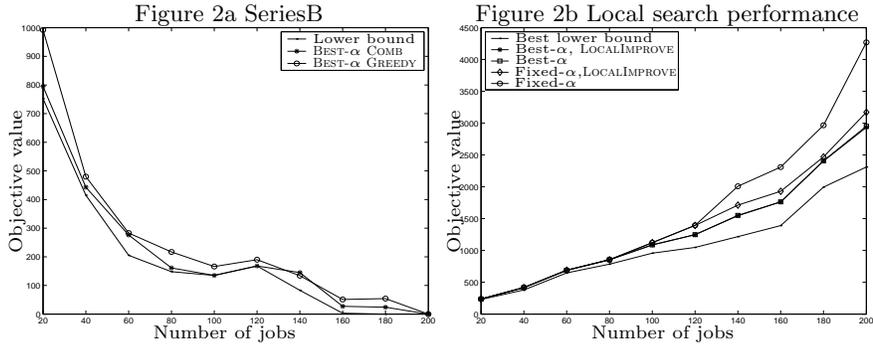
horizon and processing time of each job. We discuss the results of this small test in Section 5.

## 5 Results

We use results from the three data series described in Section 4 to illustrate the effectiveness of the algorithms, since the results from those series are representative of all the problems that we generated. Because of limited space, we focus mainly on the common conclusions, instead of analyzing in detail any specific results for the algorithms.

We ran GREEDY on a 1.8 GHz Pentium 4 with 512 MB of RAM running Linux kernel 2.4.7-10. The SPECint2000 and SPECint-base2000, performance comparisons are, respectively, 619 and 599. CPLEX 9.0, PICO, and COMB ran on a 3 GHz Xeon with 2 GB of RAM running Linux kernel 2.4.20-18. The SPECint2000 and SPECint-base2000 performance comparisons are, respectively, 1138 and 1089. Because of the considerable difference in speed between the two machines, we only analyzed runtimes for CPLEX 9.0, PICO, and COMB. More information on the SPEC performance comparison benchmarks can be found at <http://www.specbench.org>.

Figure 1 shows the performance of the heuristics BEST- $\alpha$  and GREEDY as a percentage of the objective value above the best known lower bound. In most cases this lower bound is the optimal integer solution. However the most difficult problems (SeriesA with 160,180 and 200 jobs, SeriesB with 160,180 and 200 jobs) were too big to be solved optimally in a reasonable amount of time and for these we give the best lower bound encountered during the branch-and-bound computation. Figure 2a shows the performance of BEST- $\alpha$  and GREEDY along with the best known lower bound for SeriesB. Since the last problems of SeriesB



**Fig. 2.** Performance on SeriesB and the effect of local search. The graph on the left shows the results for SeriesB. The lower bound is the optimal IP solution for 20 – 140 jobs problems. Since the lower bound is zero for the 160 – 200 jobs problems, we show the actual objective values for all the algorithms and the lower bound. The graph on the right shows The effect of LOCALIMPROVE on BEST- $\alpha$  and FIXED- $\alpha$ , using GREEDY.

have lower bound of 0, we cannot find the percentage over the best known lower bound for these problems. Figure 2b shows the effect of using LOCALIMPROVE with the heuristics.

In Series A, CPLEX required seconds for the 20, 40, and 60 job problems, minutes for the 80 and 100 job problems hours for the 120 and 140 job problems and could not solve the 160, 180, and 200 job problems in under 48 hours. In Series B, CPLEX required seconds for the 20, 40, 60, 80, and 100 job problems, minutes for the 120 and 140 job problems, and did not solve the 160, 180, and 200 job problems in under 48 hours. CPLEX solved the 20 to 160 job problems of Series C in at most 72 seconds and required several minutes for the 180 and 200 job problems. CPLEX’s success with the Series C problem was related to the fact that it did not require any branches in the branch and bound tree - the optimal solution to the linear program was integral. The heuristics ran on all problems in Series A in under 133 seconds, in Series B in under 321 seconds, and under 130 seconds for all problems in Series C.

**BEST- $\alpha$  is better than FIXED- $\alpha$ .** In almost every instance where both heuristics do not achieve the optimal solution, BEST- $\alpha$  yields significant improvement over FIXED- $\alpha$ . Figure 2b shows this for SeriesA; the results for other series were similar. The difference increases as the problems become more difficult. In the extreme case the solution from FIXED- $\alpha$  without LOCALIMPROVE is more than factor of 4 times the solution from BEST- $\alpha$  without LOCALIMPROVE.

**COMB performs better than GREEDY.** For SeriesA, COMB was on average 14.3% over the optimal solution, compared to 16.2% for GREEDY. For SeriesC, the average was 10.5% for COMB and 14.5% for GREEDY. Of the 10 instances in SeriesB, COMB performed better in 8 instances, while GREEDY only performed better in only one instance. This can be seen in Figures 1 and 2a.

**LOCALIMPROVE helps FIXED- $\alpha$  but it does not help BEST- $\alpha$ .** Figure 2b shows the effect of LOCALIMPROVE for SeriesA. The results for other series

were similar. For  $\text{FIXED-}\alpha$ ,  $\text{LOCALIMPROVE}$  has a significant improvement on the solution, while the effect on  $\text{BEST-}\alpha$  is minimal. Of the 30 instances, only 3 instances had improved solutions when using  $\text{BEST-}\alpha$  with  $\text{LOCALIMPROVE}$  compared to using only  $\text{BEST-}\alpha$ . Also while  $\text{LOCALIMPROVE}$  significantly improves the solution of  $\text{FIXED-}\alpha$ , there was not a single instance where  $\text{FIXED-}\alpha$  with  $\text{LOCALIMPROVE}$  found a better solution than  $\text{BEST-}\alpha$  without  $\text{LOCALIMPROVE}$ .

**A small decrease in accuracy gives a huge decrease in running time.**  $\text{CPLEX}$  required several hours to solve the most challenging tractable problems in these series. Moreover, the most difficult problems ran for more than two days without a solution. The heuristics scale efficiently with the problem input size and run in minutes at worst; we get a solution to the most difficult problem in under 7 minutes. In addition, the heuristics usually return a solution that is within 10% of the optimal value.

**The best heuristic is  $\text{BEST-}\alpha$  COMB.** Overall, our results indicate that  $\text{BEST-}\alpha$  and COMB is the best combination.  $\text{LOCALIMPROVE}$  does not yield any significant improvements on the  $\text{BEST-}\alpha$  and its running time hardly justifies its inclusion. Hence the best combination in terms of speed and results is  $\text{BEST-}\alpha$  COMB.

We tested  $\text{BEST-}\alpha$  GREEDY on our full-year problem and found that it computed a solution within 14% of the linear programming relaxation lower-bound. Based on the results of the the other experiments, we hypothesize that  $\text{BEST-}\alpha$  COMB would perform well on the larger problems as well.

## 6 Conclusion and Future Work

We have provided evidence that sophisticated algorithmic ideas, designed for combinatorial scheduling problems and designed with approximation algorithms in mind, can be used in heuristics to obtain good solutions to a difficult, “real-life” project scheduling problem. In addition to providing solutions in a fraction of the time of exact methods, our heuristics allow a finer granularity of time.

Our experiments also provide evidence that  $\alpha$ -points can yield reasonable approximations for a difficult project scheduling problem. This supports the idea that  $\alpha$ -points can be practically used in several problem contexts. Our experiments demonstrate that, for this problem,  $\text{BEST-}\alpha$  is a significant improvement on  $\text{FIXED-}\alpha$ , and multiple choices for  $\alpha$  should be used in any algorithm employing  $\alpha$ -points. In particular,  $\alpha$ -points provide an inexpensive way to generate many feasible schedules from one linear program. We have also demonstrated that  $\alpha$ -points can be used in situations with hard deadlines.

In the future, we plan to run our heuristics and exact methods on full-scale data. We would like to compare the overage of an optimal plan to the overage for an optimal true schedule. We wish to quantify the effect of improved heuristics, and of rerunning heuristics on multiple search-tree nodes, on running times of exact methods. Additionally, we would like to consider different priority rules, especially some suggested for the resource leveling objective by Neumann and Zimmerman[11]. We are particularly interested in testing greatest resource demand or greatest resource demand per time unit. Finally, we would like to in-

investigate the use of start-time based  $\alpha$ -points further, both experimentally and theoretically.

## References

1. P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal on Operations Research*, 112:3–41, 1999.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill, second edition, 2001.
3. Jonathan Eckstein, Cynthia A. Phillips, and William E. Hart. PICO: an object-oriented framework for parallel branch and bound. In *Inherently parallel algorithms in feasibility and optimization and their applications (Haifa, 2000)*, volume 8 of *Stud. Comput. Math.*, pages 219–265. North-Holland, Amsterdam, 2001.
4. M. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 591–598, 1997.
5. Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang. Single machine scheduling with release dates. *SIAM J. Discrete Math.*, 15(2):165–192 (electronic), 2002.
6. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, August 1997.
7. Edwin A. Kjeldgaard, Dean A. Jones, George F. List, Mark A. Turnquist, James W. Angelo, Richard D. Hopson, John Hudson, and Terry Holeman. Swords into plowshares: Nuclear weapon dismantlement, evaluation, and maintenance at pantex. *Interfaces*, 30(1):57–82, 2000.
8. F. Margot, M. Queyranne, and Y. Wang. Decompositions, network flows and a precedence constrained single machine scheduling problem. talk by M. Queyranne at IMPS 97, 1997.
9. R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.
10. Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Resource-constrained project scheduling: computing lower bounds by solving minimum cut problems. In *Algorithms—ESA '99 (Prague)*, volume 1643 of *Lecture Notes in Comput. Sci.*, pages 139–150. Springer, Berlin, 1999.
11. Klaus Neumann and Jürgen Zimmermann. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal on Operations Research*, 127:425–443, 2000.
12. Cynthia A. Phillips, R. N. Uma, and Joel Wein. Off-line admission control for general scheduling problems. *Journal of Scheduling*, 3(6):365 – 382, 2000.
13. Martin W.P. Savelsbergh, R.N.Uma, and Joel Wein. An experimental study of LP-based scheduling heuristics. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 453–461, 1998.
14. A. S. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. In R. Burkard and G. Woeginger, editors, *Algorithms – ESA '97*, volume 1284 of *LNCS*, pages 416 – 429. Springer, Berlin, 1997. Proceedings of the 5th Annual European Symposium on Algorithms.