

Travel Time Optimization **for** **Public Utility Job Scheduling**

Jacob Feldman, IntelEngine, CTO
(732) 287-1531, feldman@ilog.com

Nicholas P. Sekas, LILCO, Senior Management Consultant
(516) 391-6729, nsekas@lilco.com

Abstract

ILOG Solver and Scheduler have been used to develop two scheduling engines for Long Island Lighting Company (LILCO). Both engines assign limited resources (derived from humans, vehicles and equipment) to various public utility design & construction maintenance jobs. A key challenge has been to create a schedule which minimizes travel time between job locations while simultaneously balancing other scheduling objectives such as: schedule jobs as soon as possible, use the least costly skills/vehicles/equipment, and levelize resource utilization. Resulting schedules should also respect user-defined scheduling rules and try to honor user-defined preferences. Several “heuristic” approaches to this multi-objective problem were applied within the two LILCO scheduling engines. This article provides a detailed description of these approaches while illustrating ILOG Solver/Scheduler implementation concepts.

LILCO Resource Management System

The Long Island Lighting Company (LILCO) provides electric and gas service to more than one million customers on Long Island, New York. LILCO's 5000+ employees cover a service territory of 1,230 square miles. The Corporate Resource Management System (CRMS) is a work planning and scheduling system which LILCO uses to schedule the field design and construction maintenance work of its Electric, Gas and Fossil (power generation) business units. Based on ILOG scheduling engines, CRMS represents a huge step forward by utilizing constraint-based technology to: improve work management, meet customer needs, reduce costs, and become more competitive.

Basic Scheduling Methodology

LILCO built CRMS to optimize its scheduling process. LILCO's legacy Management Information System (MIS) provides most of the information needed for scheduling, however, several other legacy systems serve as source data for the CRMS database. MIS job information includes the job's earliest possible start date, latest possible finish date, corporate priority, the estimated duration of the job in work hours, and the resource pattern. This last item is the minimum mix of people skills, vehicles and equipment needed to efficiently complete the job.

CRMS is designed to consider these attributes and to find dates & times within the window bounded by the earliest start date and the latest finish date during which all the needed resources are available. It assigns resources to the job and ensures that these resources cannot be claimed by another job within the same time frame. Other jobs requiring the same resources are scheduled adjacent to the first job so that travel time is minimized. For a single operating area's schedule, it could take an enormous amount of time to sift through hundreds of jobs and resources, and to then intelligently assign each job a start time based on priorities and resource availabilities. Taking into consideration that users (scheduling analysts) want to "play" with different schedules to find the most appropriate one, a major concern was that CRMS might be unacceptably slow.

Most "off-shelf" public utility scheduling systems are in reality bookkeeping systems which track the use of resources as they are assigned to work by human schedulers and manage the relationships among related tasks. While these systems are very helpful in identifying and resolving resource and job conflicts, they do little *actual* scheduling of the work. In LILCO's language, "schedule" means to find and assign available resources to each job and to pick job start and finish times which honor various customer commitments and other requirements. Minimization of travel time is one of the most important scheduling objectives.

Conceptually, CRMS has two major parts. The first is comprised of the graphical interface to the users and all the software required to gather and disseminate information from/to the legacy systems which provide the raw data needed to schedule work. The second part is the unique heart of CRMS. This part is a constraint-based environment that could be created and modified by scheduling engines.

Each Scheduling Engine (SE) solves a specific scheduling problem sharing the same scheduling environment. An engine receives job importance characteristics (i.e. priority, due date, early start date), requirements (resource pattern) and resource availability (vacation/sick/training/meeting and truck maintenance/repair unavailability). A user can define specific parameters and optimization criteria via the CRMS graphical user interface (GUI). The engine usually defines scheduled start dates and times for several or all jobs by assigning required resources to them. The SE tries to "honor" user requests while searching for the best solution using different scheduling strategies.

Some of the scheduling requirements are rules and thus, must be satisfied for a job to be scheduled while others are preferences and *should* be satisfied if possible. It is important to remember that the user always "has the last say." The user may have timely "real world" information that the system is simply unaware of. As such, the SE allows the user to manually override any individual scheduling decision at any time. An example of a strong user preference is when the user "freezes" a low priority job "in time" so that it will be scheduled tomorrow as opposed to next month since it was promised early to the customer. In this example, the SE will internally give this job a much higher value and try to schedule it much earlier than usual; thus, dramatically increasing its chances of becoming a scheduled job.

As users have seen what real world situations can be supported by the scheduling engines, they want more and more until some concessions ultimately must be made. We have learned, with real world experience, that several business rules that seem so innocent and easy to implement on the surface, turn out to be in *conflict* with one another when combined. Because all user needs frequently could not be satisfied all of the time and some sacrifices are necessary, a balanced schedule becomes the ultimate goal. This has led LILCO to build a family of scheduling engines that can flexibly respond to different customer needs depending on a variety of scheduling situations.

LILCO has developed and is currently using two highly optimized and customizable scheduling engines in production mode: Construction Scheduling Engine (CSE) and Designer Scheduling Engine (DSE). Both engines share a library of specialized C++ components and patterns that are considered as building blocks for new scheduling engines.

Construction Scheduling Engine

The Construction Scheduling Engine (CSE) was the first engine developed for LILCO. This engine is currently scheduling all construction workers in the Electric and Gas organizations within the Company. The Fossil organization (power plants) is in the midst of its production roll-out. The CSE is unique in that it assigns “crews” made up of several employees, vehicles and equipment to jobs on a daily basis. Most jobs are short in duration (under 8 hours), so minutes granularity is appropriate. The CSE is given information about each job that assists it in producing a schedule. With the limited resources, its main function is to intelligently allocate resources to those jobs that are most valuable to the Company first. The CSE can handle multi-shift jobs, does resource reservation for emergency jobs, assigns backup jobs, has a sophisticated mechanism to handle jobs in progress and frozen jobs, and more (see details in the First ILOG Solver/Schedule User Conference Proceedings: July, 1995).

The CSE has a long list of scheduling rules and preferences. These characteristics could have different values. The ILOG-based implementation allows additions to these lists without changes in the search algorithms. More scheduling rules and preferences can be added and their values can be redefined by the user for *each* scheduling run.

Designer Scheduling Engine

The Designer Scheduling Engine (DSE) pursues a very different approach to scheduling. The DSE schedules “1-person” crews made up of design engineers who “design” or “spec out” jobs so that they can be handed off to the construction crews who actually lay the pipe in the ground, set poles/transformers, etc. Since designers have office and field components to most of their jobs, an accurate estimate of each job is extremely difficult to supply. In addition, many external and uncontrollable factors exist (such as field permits, traffic rules/regulations, etc.), so it’s fruitless to ask the DSE to produce a highly granular schedule where jobs are scheduled minute to minute. Instead, the DSE produces a weekly schedule for each designer in the given scheduling domain where job importance, skill matching efficiency and travel optimization play large roles in determining job assignments. The DSE has a several specific features that make it different from the CSE:

- 1) Entire job durations are not scheduled all at once from beginning to end; they are “sliced” up over early start to late finish end dates.
- 2) Schedule is based on weeks, not individual days and times.
- 3) Users take engine results only as a “suggestion” and create a committed schedule per designer from there.
- 4) Jobs are *gradually* committed to designers as other jobs are completed in the field.
- 5) “Warns” users when a job’s due date is getting close, when a suggested job has not been committed and it’s getting late, etc.

- 6) Highly flexible via a set of scheduling parameters that can be customized for CRMS job domains. A “common sense” cost approach is used in assigning the best possible candidate to each job.

Parameterized Scheduling and Resource Allocation

Both scheduling engines (CSE and DSE) are constantly being run by users with different needs and certainly, different data characteristics. For example, some users may prefer to produce very travel-friendly schedules, while other users may “weaken” the weight of travel since most of their jobs densely populate a small geographic area. As such, we cannot have a fixed travel cluster or radius. Such a value should be “dialable” per scheduling domain.

In general, the business model should be clearly separated from the search routines, so that changes to the business can be easily implemented. For instance, if union negotiations generate a different allowance for coffee breaks, this change can be easily implemented at a global level. On the other hand, if new government regulations change the emphasis on commercial versus residential customers, the search routines may be changed to value one over the other when scheduling customer work. The benefit is lower system upgrade and maintenance costs, and the ability to provide system support for a rapidly changing business.

To customize scheduling and resource allocation strategies for different scheduling situations, a SE uses configuration parameters specified via the GUI or a configuration file. There are special priorities between different scheduling parameter sources:

- run-time parameters;
- user-specific configuration file;
- domain-specific configuration file;
- environment variables;
- hard-coded defaults.

In this article, we consider mainly travel-related parameters. Most of them specify the weights and values that can dramatically change the search strategy. In general, we can classify these parameters in three categories:

- scheduling parameters;
- resource allocation parameters;
- scheduling versus resource allocation parameters.

Basic scheduling logic is very often returned to as a means to treat specific job comparison situations. Scheduling parameters allow an SE to find the next job to be scheduled. We associate an internal “job selection cost” with each job. The smaller a job’s cost, the more preferable the job is. Scheduling parameters such as “weight of priority” and “weight of late finish date” allow an SE to recalculate job selection cost when considering which job to schedule first.

For resource allocation, the SE often has to make a choice between several resource-candidates to be assigned to the current job. As such, we also associate an internal “resource selection cost” with each resource. The smaller a resource’s cost, the more preferable the resource is. Resource allocation parameters such as “weight of travel”, “weight of skill matching”, and “weight of resource levelization” allow the SE to recalculate resource selection cost when considering which resource-candidate to allocate first. In short, a “low cost competition” develops while the SE tries to schedule each and every job.

And finally, there are a lot of situations where the SE has several scheduling solutions to choose from. Here, it must choose which parameters are more important: scheduling weights or resource allocation weights. For example, the SE can schedule a job as soon as possible but one downside may be that the assigned resources may have to incur extra travel time or they may be slightly over-qualified for the job. In contrast, the SE can opt to schedule the job later in an effort to assign more preferable resources. The parameters specifying such preferences belong to the third category.

For travel time optimization, we applied two different approaches and used a set of scheduling and resource allocation parameters for our main engines. The DSE uses an “allocate resources to jobs” approach while the CSE uses an “allocate jobs to resources” strategy. Although one may think that it should be easy to select just one “best” approach (either CSE or DSE travel algorithm) and apply it to both engines, we have found that given the two drastically different scheduling environments of design vs. construction scheduling, each approach suits each engine just fine.

Assigning Resources to Jobs: The DSE’s approach

All jobs are scheduled according to job selection criteria. In LILCO’s case there are two major job “importance” factors: job priority and late finish date (LFD). To satisfy our travel optimization objective, the DSE never violates job ordering. Instead, it looks for the lowest cost resource that can be assigned to the next most important job. The DSE defines a cost associated with each designer-candidate as a scalar product of values associated with different designer characteristics. Lowest cost is defined from several business-associated cost components that are described below.

Let's suppose the DSE currently schedules a job "JOB" that requires a skill "SKILL" in an operating area "AREA" at the job's location "LOCATION" during DURATION hours from WEEK-1 to WEEK-N. First, a designer-candidate should possess the "SKILL" (directly or through skill replacement rules defined by the user) and should be available during the job's required hours. All other designer-candidate characteristics are scheduling preferences (not rules) and depend on the current assigned workload of the designer and the user-defined weight-parameters. The initial cost of each designer-candidate is equal to zero. Once the DSE identifies all of the designer-candidates for the current job, a "competition" begins to determine who the best candidate will be. Furthermore, the DSE then must try to assign the best candidate to the current job. If it fails, it will then move on to the next best candidate and try to assign this one to the job, and so on, until the job is placed in the scheduled or unscheduled status.

Resources Inside Job Cluster

The JOB cluster can be defined as a geometric area within CLUSTER_DISTANCE from the LOCATION. A designer-candidate may already have been assigned by the engine or manually frozen to some jobs inside JOB's cluster. We will reference such jobs as "nearby" jobs. For every nearby job, the DSE calculates "nearby time" as a sum of all time intervals when this job intersects in time with the JOB's time intervals. This intersecting knowledge is very important. For example, if the DSE finds that designer Tom is assigned to a lot of work within the current cluster, but Tom does not start on this work until 3 weeks from "today" - then the DSE should be aware of this because the current job's entire span may only be from today until sometime next week. Hence, if there is no intersection of job spans, then why should the DSE concern itself with assigning the current job to designer Tom?! Finally, the DSE calculates a "total_nearby_time" per designer-candidate and uses this number in combination with user-defined weights to decrease the candidate's total cost.

This approach allows the DSE to group all resources by geographic areas to minimize travel time. It is a form of implicit "density" of designer-job pairs surrounding the current JOB. A job's cluster can be thought of as a circle, whose initial distance is defined by the user, where the job's exact location is the center of this circle. Any designer-job pairs within this circle are considered to be part of the "nearby" family and are heavily weighted within the DSE's travel optimization cost structure.

Resources Outside Job Clusters

During scheduling, the DSE may be faced with a situation where it has two designers to choose from which are very close in terms of best choice. Assume both designers are equally qualified to perform the current job's requirements and that both designers already have the same number of jobs in the current job's vicinity (cluster). Another piece of knowledge that would help the DSE here would be to "look outside" the current

job's neighborhood to examine if either of these two designers have job assignments elsewhere and to what extent. Here, the DSE wants to see outside the current cluster so that it does not make the mistake of "stealing" a designer who is already working on a lot of jobs in another neighborhood!

For example, suppose the first designer had 4 jobs within the current cluster and 1 job outside while the second designer had the same 4 jobs within but 5 jobs outside. To a human with a top-down or "bird's eye" point of view, this decision would be simple: assign the current job to the first designer. However, the DSE needs this "outside cluster" knowledge to reach the same decision.

The DSE calculates the total time outside the cluster to increase the cost associated with this designer-candidate and thus, to make this designer less preferable. However, because the "outside time" should serve only as tiebreaker and is not as important as "inside time", the DSE does not use the absolute value of "outside time." It calculates the relative percent that the "outside time" is of the designer total available time (capacity). And only this percent will be added to the designer cost.

The real implementation has several more parameters for even more flexibility, but once again, the DSE relies on simple heuristics that indirectly or directly relate to the goal of optimizing travel. This heuristic only "gets applied" when it makes sense to do so, just as a human would rationalize any scheduling situation.

Idle Resources

At any scheduling point, a resource can be idle or still remain unassigned to any jobs. In LILCO's situation, idle resources are sitting at headquarters with known locations. To take this resource state knowledge into consideration, the DSE calculates "inside" and "outside" times for each idle resource and decreases or increases the cost associated with this resource correspondingly. In many cases, we have found that it is more travel-efficient to send an idle designer from headquarters to a job, instead of pulling another designer away from his/her increasingly concentrated neighborhood. Hence, this idle resource knowledge is available to the DSE. Only if it makes sense to "listen" to this knowledge, it will do so.

Unavailability Tolerance and Variable Slicing

From a business perspective, LILCO knows that design work (DSE scheduled) estimates are not set in concrete. Due to the office and field components and several external factors out of the assigned designer's control, many design jobs turn out to have actual scheduled hours far different from the original estimate or job duration "fed" to the DSE. In addition, the larger the design job duration, the more flexibility the DSE should have since volatility of "actuals versus estimate" only increases.

Many jobs have short “widths” or “spans” from early start date (ESD) to late finish date (LFD). Typical jobs are usually less than or equal to 7 days “wide” and include total job duration of 4-8 hours. Scheduling these jobs is relatively easy. Just find the lowest cost resource and assign it to the job at some scheduled start/end dates within the job’s span. However, other jobs include “wide” spans of weeks and months, and very often, larger job durations. Here, since we are dealing with a wider scheduling period which very often has real world obstacles in the way of contiguity, we need to become more flexible.

Consider the following job:

ESD=during week 1, LFD=during week 4, Duration = 40 hours

Week	Required Hours	Available Hours	Scheduled Hours
Week 1	10	07	07
Week 2	10	20	10
Week 3	10	09	09
Week 4	10	20	10
Total	40	56	36

Even though this designer is available for 56 total hours over this job’s span, he is not available for every one of this job’s uniform weekly slices (10 hours per week). Here, the DSE uses two complementary “freedoms” in an effort to improve its travel-related decisions for this job:

1. Wide job unavailability tolerance
2. Variable slicing

The first involves the DSE recognizing the fact that the job’s ESD to LFD job span is indeed wide (assume at least 7 days). When the DSE labels a job as “wide” it is empowered to apply some user-defined degree of tolerance when assigning resources to this job. In this case, assume that the DSE was allowed to accept up to 4 hours of unavailability per resource. This allows the DSE to, in effect, schedule 36 of the 40 hours to a designer. In addition, this tolerance is calculated as a percentage of the job’s duration, so the larger the job’s duration, the more tolerance allowed. This small degree of freedom very often helps the DSE select the most travel-friendly designer-candidate for the job. And with short breaks in availability such as training/sick/vacation/meeting/etc., this logic frequently improves the schedule and, more importantly, is consistent with the way business is conducted in the real world.

So, if the “best” designer is unavailable for 4 hours, the designer can be “partially scheduled”. If this assignment is made, an appropriate status code will be posted to this job such that the user understands that the assigned designer is unavailable during a tolerant amount of time somewhere from the job’s scheduled start date/time to its end. Of course, the DSE always tries to schedule a job completely using 100% of the required resources’ time. This functionality is there only if the DSE needs to rely on it as a “crutch” to get a job scheduled.

Unavailability tolerance allows the DSE to use the “best” designer (from travel perspective) even in a situation where he/she is partially unavailable. Because the actual duration of a designer job often varies greatly from the original work estimate (job duration), users have empowered the DSE to “accept” limited “gaps” in designer availability in an effort to build a travel-friendly schedule.

The second “freedom” that the DSE uses to improve its travel decisions is variable slicing. Since, in the real world of design work, users do not know exactly how much time will actually be spent in the office and field, we should give the DSE some freedom when it calculates each job’s weekly slices. This freedom is measured “away” from the job’s initial uniform weekly slices. Note how in the example above, the designer is available for a total of 56 hours over the job’s span and that the job only requires 40 hours. However, in week 1 and week 3, the designer does not have enough availability to support the job’s uniform slices of 10 hours each (week 1=7 hrs, week 3=9 hrs). This is exactly where we can empower the DSE to slightly modify the uniform weekly slice requirements (up or down) in an effort to get the job on the schedule, and more importantly, to choose the best candidate.

Resource Levelization and Travel

Depending on the user, levelization of resources can be one of the most important scheduling objectives and any engine associates a relative weight to this objective. The basic requirement for resource levelization is if the engine has many resources to choose from, it should choose the one with the least assigned work. In general, the higher WEIGHT_OF_LEVELIZATION, the more "leveled" the work distribution will be. Usually, the engine adds to a candidate's cost a value:

$$\text{WEIGHT_OF_LEVELIZATION} * \text{current_usage_percent}$$

where current_usage_percent is based on an individual resource’s total available minutes from today to the extended horizon. As a result, the busier a resource, the less preferable it will be.

But it is not always obvious what is better: utilize all resources at 60% or utilize 60% of the resources at 100%. It depends on the management preferences and real-world situations, but it also can have a dramatic impact on travel optimization. Initially, when not too many jobs are scheduled yet, resource levelization can serve as a good tiebreaker for resources with the same (or zero) travel cost. However, when resources become more and more assigned to a limited number of geographic clusters, resource levelization may lead to too many formed clusters and, as a side effect, a dramatic increase of travel time.

To minimize this undesirable effect and still honor resource levelization, a user can set WEIGHT_OF_TIME_UTILIZATION to a weaker value than the weights more closely related to travel optimization. The DSE, for example, takes one more step and calculates a “usage-percent” for each designer as a number of used days relative to a number of available days. So, some “start-up” tweaking of weights has been our experience when a new scheduling domain comes online. Depending on the scheduling environment of jobs and resources and the user’s optimization preferences, many weights will be dialed up and down until a balanced schedule is achieved.

Skill Matching and Travel

Skills are typically hierarchical in nature. For example, a senior designer possesses the skills of a regular designer and can be used as his/her replacement. If Skill-C can be replaced by Skill-B and Skill-B can be replaced by Skill-A (so, Skill-A can do the work required by Skill-B and Skill-C), then the DSE will consider a designer-candidate with Skill-A as more “costly” than one with Skill-B or Skill-C for a job that requires just Skill-C.

If the designer possesses the current job’s exact skill requirement, his cost, with respect to the “skill matching” cost component, will not change. If the designer possesses a skill that is “above” the current job’s required skill, then his cost will be increased by the value:

$$\text{WEIGHT_OF_SKILL_MATCHING} * \text{skill-distance}$$

where skill-distance is the shortest distance from the designer’s skill set to the job’s required skill on the resource replacement tree.

As in resource levelization, the skill-matching heuristic can have a negative impact on travel time optimization. To avoid this, in most situations a user sets a much lower value for WEIGHT_OF_SKILL_MATCHING than the “direct” travel related weights.

“Minimize Travel” vs. “Start ASAP”

Another example of conflicting business objectives is “minimize travel time” versus “schedule job as soon as possible.” All jobs have early start dates (ESDs) and late finish

dates (LFDs). Within this job span, a duration exists which denotes the amount of work estimated to be performed. An example of this is a job with ESD=July1, LFD=Aug1, and duration=16 hours. Here, a typical uniform weekly slice distribution may be 4 hours per week over an approximate 4 week span from July 1 to August 1. The “wider” this ESD-to-LFD span, the more “room” the DSE has to bend some rules in order to optimize the schedule, especially with respect to travel.

Our experience has been that the “current” week is always work intensive. This is because the current week always has its share of emergency or “fire” jobs, which absolutely need to be scheduled due to customer commitments. In addition, jobs are generally added to the backlog in a trickle fashion. Very often, we don’t know about a job until a few weeks before its due date. So, getting back to our above example, if we empowered the DSE to push out some of week 1’s 4-hour slice to weeks 2-4, then we may improve our travel decision.

How? Suppose designer Bob is the best travel-candidate for the job since the job is located near most of his other job assignments (the current job is in Bob’s neighborhood). However, due to resource availability constraints (i.e. 40 hours per week), Bob is already fully utilized during week 1. This obstacle will cause the DSE to overlook Bob as the best candidate; causing a poor travel-related scheduling decision. However, if the DSE was allowed to “push out” this job’s scheduled start date to week 2, Bob could be chosen; creating a travel-friendly work schedule. The DSE uses this flexibility only when it needs to - in an effort to optimize travel decisions throughout its scheduling run. In addition, note how this concept of empowering the DSE to push out a job’s scheduled start date is very similar to how we empower the DSE to vary the initial uniform slicing (see earlier section on *Unavailability Tolerance and Variable Slicing*) - both in an effort to optimize travel-related decisions.

Incremental Clustering

If an engine uses a fixed cluster or radius, it can overlook important jobs that are just outside of the cluster border. A job may have zero “nearby” jobs in the 5-minute cluster and 10 nearby jobs in the 6-minutes cluster. When we approach a new job for scheduling, we want the engine to “look around” to get an idea of what the situation of job assignments are around the current job. To achieve this, the DSE examines a new job’s surrounding designer-candidate job assignments using an initial, small cluster (i.e. 5 minutes). If there is not sufficient information within this cluster to make an informed decision, the DSE will “step up” it’s cluster size by a user-defined number of minutes in order to “learn more” about the surrounding job assignments and locations. The DSE will continue to expand the size of the current job’s cluster until it has enough information to “try” a “best” choice designer for the job, with respect to our resource specific optimization criteria: travel, skill matching, resource levelization, etc. Sometimes, the job density surroundings may be so sparse that the DSE will not deem it

wise to include travel in its decision. Here, other user-defined weights will become more powerful. Lastly, as the DSE moves on to schedule the next most important job, it builds a new cluster with that job's location serving as the new reference point or center of the new cluster.

Thus, we have developed "incremental" clustering logic which allows the DSE to absorb as much information as it needs to make an intelligent decision related to travel. From experience, we found that the DSE was making some hasty, shallow decisions on certain jobs. Blanket-type or "one size fits all" optimization approaches was, in hindsight, an over-simplified approach which produced sub-optimal schedules for users. The incremental clustering logic minimizes these poor decisions. As a technical bonus, this approach has proven to be very efficient from a CPU processing perspective. The DSE only increments its vision or cluster size when it needs to. Being developers, we however are proud to tell that this logic had been proposed by users themselves. Thus, our general principle "keep users involved as much as possible" brings fruitful results.

Assigning Jobs to Resources: The CSE's approach

The CSE uses an alternative approach than the DSE approach in trying to minimize travel time. Instead of looking for the best resource in the vicinity of the current job, the CSE looks for the next "most important" job to which the previously scheduled "crew" can be assigned. The CSE uses "job-candidates" instead of "resource-candidates" and compares them based on similar optimization objectives.

Daily Job Clustering

As we mentioned above, for construction scheduling, travel time optimization is extremely important but only during one working shift (since each crew "resets" each day by starting out from headquarters again). All jobs are still scheduled according to the CSE's main job importance ordering selection criteria (priority, late finish date, early start date, etc.). However, to satisfy the travel optimization objective the CSE is empowered to violate this job importance order in an effort to assign the current crew to the closest job. In addition, the CSE is smart enough to relax its "find closest job" rules when a job that must be done today/tomorrow is located within the current cluster.

The CSE dynamically builds each job cluster as it tries to schedule jobs. Once a team of resources is built and assigned to the current job, the CSE looks at other nearby jobs that this team is qualified to perform during that same day or shift. In general, if several jobs are "close" in proximity and share the same or similar resource pattern demands, they can belong to the same job cluster where travel time will play a significant role in the scheduling of each job.

Team Saving/Splitting Constraints

Teams are assembled from a pool of resources from a specific site, which is called an area. Once a team is assigned to its first job, the CSE then tries to assign this team to a second job, which requires the same or similar resource pattern. However, if no other job requires the same or similar resource pattern for the day, the team may be split to satisfy other job demands. The best case scenario is when a team is split into exactly two or more teams.

The underlying team splitting rule states that when a team is "formed" during the day, it must be formed solely from another team. As such, a team becomes a kind of "mini-pool" of resources. For jobs which are not subject to these geographic restrictions (i.e. power plant jobs), the "team splitting" rule still applies. Here, we benefit in that if a team "overruns" its estimated hours, it creates less havoc in the operation. That is, since the team is usually re-assigned as a whole, the entire team is either late or on time for subsequent jobs.

Consider an example of Job A with priority 20 and resource pattern equal to five Linemen and two 50-ft bucket trucks. If this job ends during the day and the CSE finds a new job, say Job D, with the same resource pattern and priority 50, the CSE will schedule this same team to Job D. However, if the CSE finds two other jobs that are more "valuable" than Job D, then it will "split" Job A's team to schedule these jobs first. Assume Job B has a priority 25 and resource pattern equal to three Linemen and one 50-ft bucket truck. Also assume Job C has a priority 28 and resource pattern equal to two Linemen and one 50-ft bucket truck. Lastly, assume that the CSE could not find any other resources to satisfy the demands of Job B & C. Therefore, for jobs B & C to be scheduled, Job A's team must be split. Here, it is more valuable for the CSE to schedule Jobs B & C at the expense of Job D. A side effect of this kind of decision is once again when the CSE tries to minimize travel time!

The CSE has *resource pattern hierarchy* knowledge during scheduling. A situation may arise during scheduling where the CSE must decide which user request to honor: 1) assign crew to job far away that requires the same exact skills, vehicles, and equipment, or 2) assign crew to nearby job that is slightly over-staffed and/or over-equipped for the job. We have found that users prefer to choose option #2, provided that they limit the CSE to reasonable over-staffing and over-equipping. Here, we again have control over how much flexibility that the CSE uses via dialable parameters (i.e. HUMAN_SHORTAGE=1, VEHICLE_SHORTAGE=1). Resource pattern hierarchy knowledge allows the CSE to recognize bigger decision making factors when building the schedule. Having this knowledge allows the CSE to use it when it is relevant to the current scheduling situation.

Solving Traveling Salesperson Problem

The general purpose of this function is to minimize travel time for the same team during one workday or shift. The CSE solves the Traveling Salesperson Problem (TSP) in post-processing mode when all teams and their daily assignments are already known. The search algorithm for this TSP follows main ideas from the earlier ILOG Solver user manual.

Dynamic Weighting

A real-world experience leads us to a provocative conclusion: each set of data should really have its own customized scheduling strategy!

Many user domains (even in the same organization) have extremely different scheduling perspectives and/or data characteristics. By computing some “scheduling environment metrics” up front, the scheduling engine can determine which customized scheduling strategy to use. For example, if we know in advance that there are many important jobs which require a very high level skill and our employee resource pool includes only one of these skills, then we should choose a scheduling strategy that “reserves” this single “chief” only for jobs that require his/her high level skills. By doing this, we avoid “wasting” this resource’s time on lower skilled jobs, even if it means incurring some other costs such as travel, work levelization, scheduling jobs later rather than earlier, etc.

Therefore, we cannot expect a user to set different scheduling parameters for each scheduling run. Our scheduling engines should be “smart” enough to analyze the data during the pre-processing phase and to internally tune the scheduling parameters within user-defined limits. So, in effect, the engine “dials” it’s own weights on the fly, which it has determined as an optimal scheduling strategy for the current environment. This is an area where we will be spending much time on as the Long Island Lighting Company moves into full production mode with both scheduling engines.

Conclusion

The travel time optimization objective is closely related to several other scheduling and resource allocation objectives, and cannot be achieved with one hard-coded search algorithm. In many ways, the scheduling engines should be extremely human in their decision-making approaches. With experience, we have found that building LILCO’s multi-objective engines along with “heuristic building/tweaking” approaches has resulted in substantially higher quality schedules. Giving users the power to tune engines for different scheduling situations allowed us to convert the engines from “powerful but closed black boxes” to an open, “any solution” interface where the Company’s

employees can customize scheduling solutions by dialing in their customized business perspectives. These engines are, in many ways, communicating with human employees and gaining “experience” while consistently building more and more real-world schedules. Lastly, as LILCO gains more experience with its new scheduling and resource allocation system, we predict a real need and implementation of a family of scheduling engines for different organizational needs but similar corporate goals: to plan, schedule and work smarter everyday in order to satisfy the customer.