

LEDAS, Ltd.

A. Ershov, I. Ivanov, V. Kornienko, S. Preis, A. Rasskazov, I. Rykov

**LSE: NEW COMPUTATIONAL ENGINE
FOR LEDAS SCHEDULER 3.0
AND PERSPECTIVES OF ITS USAGE**

**Preprint
15**

Novosibirsk, 2005

This paper opens a series of publications about LEDAS Scheduler 3.0, a new version of the integrated environment for specification, modification, and solution of scheduling problems. Version 3.0 is based on the new LSE engine, which improves efficiency of solution by an order of magnitude, particularly in the most common classes of problems, due to new high-performance algorithms designed and implemented by LEDAS. The paper presents the necessary information about the components of the project: architecture, algorithms, and interface of the new engine, LSE, as well as the latest results of testing. In addition, it contains a comprehensive examination of the possibilities of integration of LEDAS Scheduler 3.0 into other products, in particular, into the systems currently under development in LEDAS, such as MORE+, a client-server system for automated meeting scheduling on conventional and mobile platforms, and a project scheduling system for the Eclipse platform with functions similar to those of Microsoft Project.

This series of preprints is published by LEDAS, Ltd., a Russian company developing and marketing intelligent software solutions.

6, Prospekt Lavrentieva, Novosibirsk, 630090, Russia.

<http://ledasgroup.com>

<mailto:info@ledasgroup.com>

Table of contents

Introduction	4
1. Computational engine LSE	7
1.1 General architecture of LSE	7
1.2 Definition of problems processed by LSE	8
1.3 Critical path method	10
1.4 Serial method	12
1.4.1 The description of serial method	12
1.4.2 An example of serial method execution	14
1.5 Serial sampling method	16
1.6 Branch-and-bound method	17
1.7 Solving of extended formulation problems	19
1.7.1 Support of due dates	19
1.7.2 Resource pools support	19
1.7.3 Solution of problems with the jobs of unfixed duration	20
2. Environment for scheduling problem resolution LEDAS Scheduler	20
2.1 Functionality of LEDAS Scheduler	20
2.2 LEDAS Scheduler architecture	22
2.3 Comparison of LEDAS Scheduler with its competitors	25
2.3.1 LEDAS Scheduler vs. MS Project	25
2.3.2 LEDAS Scheduler vs. ILOG Scheduler	26
3. Applications	27
3.1 MORE+ Meeting Scheduling System	28
3.1.1 Description of functions of MORE+	28
3.1.2 Mathematical specification of the problem in MORE+	29
3.2 Project scheduling system for Eclipse platform	30
4. Conclusion	31
References	32

INTRODUCTION

Scheduling problems can be found in almost all areas of human life. These problems may differ, depending on the application area, but all of them have much in common. The term “scheduling problem” assumes the need to arrange in time certain actions involving people, machines, equipment, and other resources; one may need to take into account their location and the required material resources, such as money, materials, etc. As a rule, problems of this kind include various natural constraints that should be met, which can be availability of participants, constraints on time, material resources, sequence of actions, and many others.

Development of software to automate solution of constraint satisfaction problems is an important activity for LEDAS. The competence of the company’s staff originates from their academic work in the 80s, and the constraint techniques for scheduling problems were implemented by LEDAS in the Scheduler project in 1999–2001. LEDAS Scheduler is an extensible environment for specification, modification, and solution of scheduling problems. Compared to Microsoft Project, LEDAS Scheduler can solve a much broader class of problems due to support for a broader set of constraints and goal functions. Compared to ILOG Scheduler, LEDAS Scheduler is better integrated, which ensures efficient storage, modification, and exchange of data with other applications by means of an extensible set of powerful tools.

In the fall of 2004, when it became clear that the market wanted such a solution, LEDAS began working on LSE, the engine for the new version, Scheduler 3.0.

The main features of LSE are:

- Dramatically improved computation efficiency, particularly for the most common types of problems, due to the high-performance algorithms developed and implemented by the company,
- Carefully designed architecture and API, allowing one to integrate LSE efficiently into other software products.

The results of the initial LSE testing have shown that on a set of several thousand tests, containing scheduling problems of 15 different classes, the new solver produces a suboptimal solution that differs from the optimal one by at most 10–15 percent. LSE is an easy to scale, high-performance tool; for problems with up to 1000 jobs, it finds a solution in less than a second, and the calculation time is proportional to the square of the problem’s size.

The LSE solver creates a foundation for a software product of a totally new class that will be made available to users for integration in the spring of 2005—initially in the LEDAS’ early access program (EAP) that has previously been tested during the development of 2D and 3D geometric solvers (see <http://lgs3d.ledas.com/download>).

LEDAS Scheduler 3.0 targets the following main large classes of applications:

- Scheduling and project management;
- Decision making support systems;
- Resource scheduling.

Along with the development of a new engine for the LEDAS Scheduler environment, LEDAS is developing, in pursuit of its own market interests, the following software components that can be integrated with version 3.0:

- Automatic meeting scheduling system MORE+ with a client-server architecture, containing clients for various platforms, including some mobile ones;
- A project scheduling system similar to Microsoft Project for the Eclipse platform (see a description of the platform at <http://www.eclipse.org>).

This paper begins a series of publications about the LEDAS Scheduler 3.0 project. Information on its main components will be provided, including its architecture, the algorithms and the interface of the new LSE engine, current testing results, as well as a description of the integration of LSE capabilities with the LEDAS Scheduler.

We begin with the terminology. Traditionally the following entities are identified in scheduling problems:

- *Task/Job/Activity* — an activity that evolves in time and is characterized by start/end times and duration.
- *Resource* — a person, machine, unit of equipment, room, etc., which can be used to perform a task, and can accordingly be assigned (partially or completely) during the performance of the task. Resources that can be assigned partially (or those which may consist of a set of small identical parts) are called *discrete*; resources that only exist as whole entities are called *unary*. These two types of resources are called *renewable*.
- *Consumable resource/Cost* — resource that can be consumed in the course of performance of an activity. Often one distinguishes cost, or consumption of money.
- *Calendar* — a chart that displays availability of resources in time or possibility of performance of a job.

- *Resource assignment* — a connection between a job and a resource that states that the resource is needed by the job. For discrete and consumable resources, it is characterized by the required quantity of the resource units. Resource assignment is the main source of constraints arising in scheduling problems. A constraint is created due to impossibility to assign the same unary resource simultaneously to two jobs; for discrete resources, a similar constraint arises if the need for a resource exceeds at some time its existing quantity.
- *Resource pool* — a set of renewable resources, each of which can be assigned to one or several jobs. This entity extends the notion of a resource and allows definition of a connection between a job and a set of resources rather than one specific resource. Different pools can overlap in an arbitrary manner, which makes this tool more flexible than unary or discrete resources.
- *Task order, precedence* — one more typical constraint occurring in scheduling problems, asserting that one job is performed earlier (*predecessor*) or later (*successor*) than another job. The constraint can link the start and end times of the two jobs in arbitrary combinations, as well as can be characterized by a positive or negative lag between the two jobs.
- *Release and due dates/Task constraints* — these introduce constraints that link jobs to specific times. They are stated in the form “start no earlier than” or “end no later than”. In what follows, the term *constraint* will be used for these constraints as well as for all other constraints typical for scheduling problems (both resource and precedence constraints).

All these entities were supported by LEDAS Scheduler 2.5 and will be supported in version 3.0. The old engine used powerful general-purpose mechanisms for solution of constraint problems (provided by the mathematical solver from LEDAS) and powerful problem specification tools enabling one to specify and solve almost any scheduling problem. Several diverse products were implemented over this engine (apart from the LEDAS Scheduler environment itself), for example, web-based meeting scheduling system MORE and staff scheduling system LETOM.

However, the LEDAS Scheduler 2.5 engine had two significant shortcomings. Firstly, being a general-purpose engine, it did not offer sufficient performance to solve the special types of problems arising in real-life applications. These problems can be of considerable size and contain hundreds, thousands, or even tens of thousands of jobs, but the constraints and links between the jobs are usu-

ally of a special nature, which allows employing specialized, more efficient constraint satisfaction algorithms. Secondly, LEDAS Scheduler 2.5 aimed at finding an exact optimal solution; according to the classical theories of mathematics, this problem is not effectively solvable. Therefore, all real software products for scheduling problems use efficient algorithms to find a suboptimal (i.e., close to optimal) solution, rather than an exact optimal one. The same approach is used in LSE, the new engine that will be used in LEDAS Scheduler 3.0.

The details of architecture and algorithms of LSE will be presented in the next chapter. In Chapter 2, the reader will find the architecture of the LEDAS Scheduler environment and the integration capabilities of this system; Chapter 3 describes the applications currently under development in LEDAS that use the computation capabilities of the engine.

1. COMPUTATIONAL ENGINE LSE

1.1 General architecture of LSE

LSE is a platform-independent C++ library whose interface (LSE API) is composed of abstract classes. This organization allows us to maximally separate the LSE interface from its implementation and guarantees that interface will be changed only when new entities or functionality appear. LSE itself consists of classes that implement the abstract interface classes and additional classes that maintain the computational part (preprocessor, validator, methods, and heuristics).

So, LSE API includes several abstract classes that present different entities — jobs, resources, constraints, a problem as a whole, a solver. Creation and deletion of objects of the classes `ILSEProblem` (planning problem) and `ILSEEngine` (solver) is performed by means of static methods; all other kinds of processing of these classes and classes presenting the problem elements is made by virtual methods. Using them, it is possible to create jobs, resources and constraints, to add them to `ILSEProblem`, to indicate the problem to be solved by `ILSEEngine`, to launch the process and obtain the results of computations. During computations, it is also possible to use a call-back function given by application in order to stop computation, to redraw interface, and so on.

The base class that controls LSE is `LSEEngine`. Its main function `LSEEngine::Run()` runs the preprocessor that performs a preliminary processing of the planning problem, for example, moves the constraints imposed on the group jobs to the level of individual jobs. Then the type of the problem under consideration

is determined and the method or a set of methods most appropriate for its solution is launched. Each method is presented by a separate class supporting, in addition to the necessary functions of data input/output, the callback mechanism that makes possible to pass control to an application or to interrupt calculations. In their work, a majority of methods use heuristics in the form of separate functions. After a method obtains a solution, LSEngine::Run() validates it with the help of a validator external to this method, and when several methods are working, a most optimal solution satisfying all the problem constraints is taken.

1.2 Definition of problems processed by LSE

At present, LSE efficiently solves the problems in the base formulation (base problems) which is an extension of the classical resource constrained project scheduling problem called RCPSP (see [1]).

The *base problem* is a quadruple $(\mathbf{J}, \mathbf{R}, \mathbf{P}, \mathbf{K})$ that has the following entities:

1. A set \mathbf{J} of N jobs with their durations defined as $d_j \in \mathbb{Z}^+$. It is supposed that the scheduling problem will be solved in a discrete time scale: $t=0, 1, \dots$. A set of jobs' starts $S = \{s_j \in \mathbb{Z}^+ \cup \{0\} \mid j \in J\}$ is called *a schedule*, and a job is considered to be active at the moments $t = s_j, s_j + 1, \dots, s_j + d_j - 1$.
2. A set \mathbf{R} of M renewable discrete resources with their quantities defined as $K_r \in \mathbb{Z}^+$.
3. A set \mathbf{P} of precedence constraints is given; each of them is specified by a quadruple (j_1, j_2, t, l) , where $\text{type} \in \{\text{'start-finish'}, \text{'start-start'}, \text{'finish-finish'}, \text{'finish-start'}\}$ and $\text{lag} \in \mathbb{Z}^+$ is a time interval (lag) between jobs. Each constraint specifies a technological job order: if, for example, $\text{type} = \text{'finish-start'}$, then j_2 cannot be started earlier than "lag" time units after j_1 is finished. ($s_{j_2} \geq s_{j_1} + d_{j_1} + \text{lag}$). The meaning of other types is similar.

Note. These constraints can be represented as weighted arcs of a job-on-node graph. The necessary and sufficient condition for existence of a schedule satisfying all these constraints is that this graph should not contain contours of a positive weight. LSE supports only acyclic graphs, which, however, is typical for the resource-constrained project scheduling.

4. The matrix K of resource assignments is given: $k_{jr}, j \in J, r \in R$. The following constraint is specified: at any time and for any resource, the sum of needs of all active jobs (i.e., jobs that are under process at that moment) should not exceed the total amount of this resource, $\sum_{j \in A(t)} k_{jr} \leq K_r, \forall t \geq 0, \forall r \in R$, where $A(t) = \{j \in J | s_j \leq t < s_j + d_j\}$ is the set of active jobs at the moment t . If the matrix (k_{jr}) is sparse (i.e., it contains many zero values), it is better to specify the assignments as a list of triples (Job, Resource, Capacity).

The following additional constraints are considered in LSE:

5. It is possible to define release dates $r_j \in Z^+$ for some jobs. A job cannot be started before its release date ($s_j \geq r_j$). Release dates are modeled by adding a dummy job of zero duration, which is a “finish-start”-predecessor of these jobs with lags r_j .
6. It is possible to define due dates $u_j \in Z^+$ for some jobs. The job for which its due date is defined should be finished before this date ($s_j + d_j \leq u_j$).

Note. The algorithms implemented in LSE give no guarantee that a schedule satisfying all constraints of this type will be composed. But this is natural, since the problem of making a schedule satisfying the due date constraints is as difficult as the problem of making an optimal schedule, i.e., it is not efficiently solvable. For problems of this type, heuristic attempts are made to meet the due dates, and the check of its satisfaction.

7. Unsimultaneous/Incompatible Jobs are defined by a list of jobs such that only one of them can be performed at any moment. This is modeled by adding a dummy unary resource and assigning it to each of these jobs.
8. Job hierarchy. Several jobs can be combined in a job of upper level. Only the jobs of the lowest level are real, the upper level jobs are intended to facilitate the project management. For example, specification of the resource or order constraints for a job of upper level is equivalent to specification of these constraints to all its subjods, respectively. The number of levels in the job hierarchy is unrestricted.

A schedule S is called *feasible (or correct)* if it satisfies all constraints 3,4,5, and 7. The problem is to find an feasible schedule such that its *makespan* (the value $C_{\max} = \max\{c_j = s_j + d_j \mid j \in J\}$) is minimal. If strict minimality of the project finishing time is of absolute priority, then we need to find an optimal solution to our problem; if we should minimize both the makespan and the problem solving time, then we search for a suboptimal solution.

Apart from the base formulation problem, below we will consider an “extended formulation” that includes jobs of two types — ordinary, with fixed duration, and jobs whose duration depends on the set and amount of the resources they use during execution.

In this case the definition of a schedule, in addition to the jobs’ starts, also includes the choice of the set and amount of resources assigned to each job. Duration of an assignment execution is defined as capacity of this assignment prescribed at input divided by the total amount of all resources allocated for this execution according to the schedule; durations of an assignment executions determine durations of unfixed jobs. Jobs with unfixed duration are divided in two types: for the first one (parallel use of resources), duration of a job is defined as maximum of durations of assignment executions in which this job appears; for the second (sequential use of resources) — as a sum of durations of executions of these assignments.

Now we return to the base formulation. As noted above, constraints 5 and 7 are modeled in terms of constraints 3 and 4. When LSE starts working, a pre-processor transforms constraints 5 and 7 to 3 and 4 and unfolds the hierarchical structure 8 of the problem.

Thus, the base formulation of the problem for LSE is reduced to the RCPSP problem, so we can use the well-known methods of RCPSP solution. Besides, we assume that, for all constraints on job order, type = FS (other types are transformed to this type by the preprocessor), and the weight of the corresponding arc in the graph is equal to lag.

Let us consider the algorithms deployed in LSE to solve base formulation problems.

1.3 Critical path method

In the case of total absence of resources (and therefore, of the constraints of type 4), the precedence graph $G = (J, P)$ contains all the problem’s constraints. In this case, the length of the optimal schedule is equal to the length of the critical (i.e., the longest) path in G . The length of the path means the sum of weights of all vertices (durations of the corresponding jobs) and all arcs belonging to this

path. In this schedule, the start time of any job is determined as the length of the critical path to this job. There exists a simple quadratic algorithm for finding such a schedule; more precisely, its complexity is $O(N^2 + CN)$, where C is the number of constraints of type 3, and N is the number of jobs.

Let us briefly describe this algorithm. A set of immediate predecessors of a job is denoted by P_j , i.e., $P_j = \{i \in J \mid (i, j) \in E(G)\}$, where $E(G)$ is the set of arcs of the graph G . The weight of an arc $(i, j) \in E(G)$ is denoted by $w(i, j)$.

1. Let $\forall j, s'_j = 0$
2. Do {
 - a) for $j=1$ to N $\{s''_j = s'_j\}$
 - b) for $j=1$ to N $\{s'_j = \max\{s''_j, \max\{s''_i + d_i + w(i, j) \mid i \in P_j\}\}$
 } while ($\exists j: s''_j \neq s'_j$)
3. $s_j = s'_j \forall j$.

The number of arc in the maximal (by this number) path to a vertex is called a *rank* of this vertex. It is obvious that the value s'_j will not be changed in the execution of cycle (stage 2 of the algorithm) for jobs that have no predecessors (of rank 0). It is easily proved by induction that for a vertex of rank k the final value s'_j will be fixed no later than after k iterations of the cycle. Since the maximal rank of a vertex in an N -vertex graph does not exceed $N-1$, we see that the number of iterations cannot exceed N . The complexity of one iteration is $O(N+C)$: to find all maxima, we need, for each vertex, to scan all arcs entering it, so, each arc of the graph is examined only once.

The resulting schedule has an interesting property: none of the jobs can be moved to the earlier period, i.e., it's impossible to lessen its start time. A schedule which has this property is called an *active* (or an *early*) schedule. An active resource-free schedule is unique and optimal, and at the same time there can exist other optimal schedules which are not early.

In case of absence of resources, LSEEngine uses the critical path algorithm and constructs an early schedule.

Along with an early schedule, the notion of a T -late schedule is also useful. If we take $T \geq T_{opt}$, where T_{opt} is the length of an optimal schedule, then we can construct a feasible schedule such that its finish time is T and none of the jobs in

it can be moved to a later period. It is clear that T_{opt} -late schedule is also optimal. The starts of the jobs belonging to the critical path in this schedule are equal to starts of these jobs in the active schedule, and for the rest s_j is strictly greater than the corresponding starts in the early schedule.

1.4 Serial method

1.4.1 The description of serial method

Now we turn to the general case when, in addition to the precedence constraints, there are resource constraints. It was shown that in this case the problem (RCPSp) is NP-complete (it includes, in particular, such well-known NP-hard class of problems as Job Shop — see [2]). So there are no efficient algorithms of finding an optimal solution to it.

But real-life applications often do not require us to find the shortest schedule of a project. It is a good result for a complex project to get a feasible schedule of acceptable length (say, 10–30% longer than optimal).

At present, there are many algorithms in this domain. Most of them are heuristic, i.e., they do not guarantee any estimates for the divergence between the solution found and the optimal one. But in practice, the average error is not greater than 10%, which is a very good result.

In LSE we have implemented one of heuristic schemes known as serial scheduling scheme (see [2]). Its distinctive feature is that it is clear and general and at the same time it achieves excellent results (in performance and optimality). This scheme in its essence is a greedy algorithm.

The algorithm execution consists of N stages; at each of them we fix the execution time for one of the jobs (i.e., after k stages we have a partial schedule for k jobs that satisfies all constraints).

We denote by S_n (a scheduled set) the set of jobs that are already fixed at the beginning of the stage n (so $|S_n| = n - 1$) and by D_n (a decision set) the set of jobs that are not fixed yet but all their predecessors are fixed (i.e., $D_n = \{j \notin S_n \mid P_j \subseteq S_n\}$). This is the set from which the job is chosen to be fixed at the stage N . This choice is made by the heuristic rule that can be defined this or that way, for reasons of a common sense. An example of such a heuristic rule is the LST-rule (latest start time): from the set of jobs D_n , we choose the job with the minimal start time in the C_{\max} -late schedule for the corresponding re-

source-free problem. This means that this job most likely has the longest chain of successors and it is advantageous to set it in the schedule as early as possible.

After choosing $h \in D_n$ we fix it in the schedule at the earliest time taking into consideration that all precedence and resource constraints should be satisfied. Obviously, such a time can be found: for example, $\max\{c_j + w(j, h) = s_j + d_j + w(j, h) \mid j \in S_n\}$ is the acceptable start time for a new job. It is also clear that a new s_h will be found in the interval from $\max\{c_j + w(j, h) \mid j \in P_h\}$ to $\max\{c_j + w(j, h) \mid j \in S_n\}$.

The heuristic rule of choice can be formalized with the help of the function $\nu(j), j \in D_n$. We choose a job with the greatest value $\nu(j)$ and with the least number j , if there are several of them. The set of active jobs of a partial schedule is denoted by $A(t)$, as usual: $A(t) = \{j \in S_n \mid s_j \leq t < s_j + d_j\}$. So, we have the following formal algorithm of the serial method:

1. $n=1; S_n = \emptyset$;
2. while $|S_n| < N$ do
 - a) $D_n = \{j \notin S_n \mid P_j \subseteq S_n\}$;
 - b) $h = \min\{j \mid j = \arg \max\{\nu(i) \mid i \in D_n\}\}$;
 - c) $t_{\min} = \max\{c_j + w(j, h) \mid j \in P_h\}$; $t_{\max} = \max\{c_j + w(j, h) \mid j \in S_n\}$;
 - d) $t^* = \min\{t \in [t_{\min}, t_{\max}] \mid k_{rh} + \sum_{j \in A(\tau)} k_{jr} \leq K_r, \forall \tau = t, t+1, \dots, t+d_h-1, \forall r \in R\}$;
 - e) $s_h = t^*; S_{n+1} = S_n \cup \{h\}$;
 - f) $n = n + 1$.

This algorithm is implemented in LSE with the quadratic complexity and is very fast — for the problems with thousands of jobs it finds a suboptimal schedule faster than in 1 sec.

Within this method, we have implemented several heuristics for choosing the next job from D_n to be scheduled. In addition to LST-rule, these are LFT (latest finish time, the job with the least finish time in the late resource-free schedule), MTS (most total successors, the job with the greatest number of all successors), and others. As the algorithm is very fast, the problem is solved with the use of all these heuristics and we take the best of all the solutions obtained.

The algorithm makes a schedule that is early one: none of the jobs can be moved to the earlier period without moving other jobs. It is easily seen that,

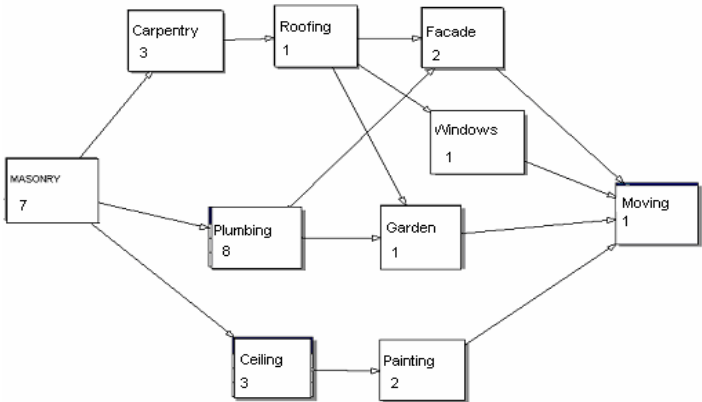
among the active schedules, there is an optimal solution to the RCPSP problem. Indeed, if we take some optimal not early schedule, we can, by a simple by-turn shift of jobs to the earlier period, obtain an early schedule whose length will be no greater than that of the initial schedule; hence, it will be optimal, also. All active schedules can be obtained if at each stage we will consider all variants of choice $h \in D_n$. Thus there exists a scheme of exhaustive search that allows the optimal solution to the problem to be found. The optimal branch-and-bound method implemented in LSE is based on this scheme. Naturally, this method is not computationally efficient (recall that our problem of a search for an optimal schedule is NP-hard). The serial method deals with only one branch of the search tree and finds one feasible solution.

1.4.2 An example of serial method execution

As an example, we consider the project of a house construction. There are 9 jobs executed in a certain order and 1 discrete resource — 2 workers (this is a reduced variant of the problem described [3]). The table below shows the precedence and resource constraints:

name	duration	predecessors	workers
Masonry	7		1
Carpentry	3	Masonry	1
Roofing	1	Carpentry	1
Plumbing	8	Masonry	1
Façade	2	Roofing, Plumbing	1
Windows	1	Roofing	1
Garden	1	Roofing, Plumbing	1
Ceiling	3	Masonry	1
Painting	2	Ceiling	1
Moving	1	Windows, Garden, Façade, Painting	1

The precedence graph is as follows (the weight of all arcs equals 0, which corresponds to a zero lag):



For the resource-free problem, it is easy to find ranks and early and late starts (jobs of the critical path are marked as bold):

Name	Rank	Early start	Late start
Masonry(1)	0	0	0
Carpentry(2)	1	7	11
Roofing(3)	2	10	14
Plumbing(4)	1	7	7
Facade(5)	3	15	15
Windows(6)	3	11	16
Garden(7)	4	15	16
Ceiling(8)	1	7	12
Painting(9)	2	10	15
Moving(10)	5	17	17

This schedule is optimal for a resource-free problem. But it is not feasible for the initial problem: at the moment $t = 7$ three jobs are being executed requiring 3 units of resource in total, which is more than available. It is easily seen that the optimal schedule of the initial problem cannot be of length 18. Indeed, the first and the last jobs are not executed simultaneously with the others, and for the rest

we have only 10 time units, whereas their total length is 21. But no more than two jobs can be executed at the same time, so, their execution cannot take less than 11 time units.

On the other hand, it is easy to construct a feasible schedule of length 19 which will be optimal. Let us consider execution of the serial algorithm using the LST heuristics: at the first stage $D_1 = \{1\}$, so job 1 is put to the earliest possible time (0). Then $D_2 = \{2, 4, 8\}$. Job 4 is chosen out of them, because it has the earliest late start, and put to the time $s_4 = 7$.

Then:

$D_3 = \{2, 8\}$ (none of the new jobs has become available yet), $h = 2, s_h = 7$;

$D_4 = \{3, 8\}, h = 8, s_h = 10$ (as soon as one of the jobs being executed is finished)

$D_5 = \{3, 9\}, h = 3, s_h = 13$

$D_6 = \{5, 6, 7, 9\}, h = 5, s_h = 15$ (job 5 — successor 4)

$D_7 = \{6, 7, 9\}, h = 9, s_h = 15$

$D_8 = \{6, 7\}, h = 6, s_h = 16$

$D_9 = \{7\}, h = 7, s_h = 17$

$D_{10} = \{8\}, h = 8, s_h = 18$

As it should be, every time s_h was chosen as the earliest time to which the job can be put with all constraints being satisfied. As a result, an optimal schedule has been constructed. Here is the illustration of this schedule (the order of construction is top-down):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
(1)							(4)											
							(2)			(8)			(3)					
														(9)		(6)	(7)	(10)

1.5 Serial sampling method

The basic method used in LSE is the sampling search method (more precisely, the multi-pass serial method — see [2]) based on the serial method. It launches the serial method a fixed number of times, each time choosing $h \in D_n$ differently at some stages and, so, scanning different branches of the tree of active schedules. We take the best of the schedules obtained. A trivial example of a choice variation is to use new heuristic at each launch, but there are not very many of them. Another way is a random choice of $h \in D_n$; in this case, to make

it more efficient, we can try to choose the most appropriate distribution of the variate. Such a distribution is called a randomized heuristics of the stochastic method. Two randomized heuristics have been implemented in LSE: first uses a uniform distribution — at each stage the choice of a job from D_n is equiprobable:

$$prob(j) = \frac{1}{|D_n|}, \forall j \in D_n,$$

and a non-uniform heuristics in which jobs with a greater value $v(j)$ are chosen with a higher probability, this way, for example:

$$\rho_j = v(j) - \min\{v(i) \mid i \in D_n\}, \forall j \in D_n.$$

$$prob(j) = \frac{(\rho_j + 1)^\alpha}{\sum_{i \in D_n} (\rho_i + 1)^\alpha}, \alpha \in [0, \infty)$$

We have an equiprobable distribution with $\alpha = 0$, a determined heuristic choice with $\alpha = \infty$, and an intermediate variant $\alpha = 1$. It was shown by testing at different problems that, for the same number of launches of the serial method, the equiprobable approach is the best. It is important that in this case the average length of the schedules is the worst, which can be explained by their higher “dispersion”. When the choice is based on the non-uniform heuristics, we scan only “good” schedules, but the best of them is worse than the best of the same number of absolutely random schedules.

It is clear that the run time of this algorithm is a certain number of times (the number of launches is specified to LSE via API) greater than the run time of the serial method. The stochastic search method achieves good results in the sense of the solution optimality. By default, 500 launches are made with the equiprobable choice and 6 launches with different determined heuristics, and the average error for about one thousand tests is 5–10%.

1.6 Branch-and-bound method

As was already mentioned, LSE deploys an algorithm that allows us to find an exact optimal solution to the problem in the base formulation. It is based on the branch-and-bound method related to the class of implicit search method (see [4]). By the estimates of specialists, now this is one of the fastest precise algorithms of solution of scheduling problems (there also exist algorithms based on the method of dynamic programming and the method of 0-1 programming, see [5] and [6]).

Two components of any method of the branch-and-bound type is the search scheme (tree) and the cutting rules (boundaries). As was already mentioned, the search scheme is based on the scanning of all early schedules, constructed by the serial method, by processing of all vertices of D_n at each stage. The number of branches in this tree is equal to the number of orderings of the set of jobs $\{1, 2, \dots, N\}$ consistent with the precedence constraints: if an arc $(i, j) \in E(G)$, then the job j should be set to the right of i . Thus, the greater is the number of the precedence constraints, the less is the number of branches in the search tree. The most difficult case for the branch-and-bound method is in the absence of the precedence constraints. In this case the search tree has $N!$ branches.

The cutting rule allows us to skip a large number of variants unproductive in the sense of optimality on the basis of information about a current record. A *record* is the best of the solutions found by this time and also we call as *record* the length of this solution.

Let the length of the current record is T . We denote by s_j^{late} the starts of jobs in the T -late schedule of the corresponding resource-free problem (in which all resource constraints are removed and only precedence constraints left).

The cutting rule is quite simple: if for a chosen job $h \in D_n$ the calculated early start time is $s_h \geq s_h^{late}$, then there is no need to complete the current partial schedule $(S_n \cup \{h\})$, because the length of any early schedule derived from this one will be greater than or equal to the record. This is a rather evident fact, since s_h^{late} is by definition the latest time when we can start the job h in order to obtain a schedule of length no greater than T (of the record) satisfying all precedence constraints. Hence, the length of all early schedules obtained at this branch will be no less than the record and would not give us a better result.

The following cutting rule is less trivial and more powerful: if $s_h \geq s_h^{late}$, then we need not complete the partial schedule obtained at the previous stage (S_n) , it means that we need not consider other variants of choice in the current D_n . This can be seen from the following observation: if at this stage we schedule another job, then h should be scheduled later and its start time can only increase in this case. So, at any branch, we come to the first case. This rule allows us to fasten the algorithm approximately twice (as compared to the first one).

Unfortunately, the complexity of branch-and-bound grows exponentially in any case and it is impracticable for the problems with more than 10–30 jobs.

1.7 Solving of extended formulation problems

1.7.1 Support of due dates

Any algorithm of solution to a problem with constraints 3 and 4 which is not exact, i.e., which finds an feasible but not optimal schedule, cannot guarantee finding even an feasible solution for the case of due dates constraints (constraints of type 6). Indeed, if the problem is such that a due date of each job is equal to the length of the optimal schedule, then any feasible schedule is optimal for it.

As a consequence, satisfaction of these constraints may be only a desirable but not necessary condition (like, for example, the makespan minimization) and should be achieved heuristically. For example, jobs with due dates, as well as their predecessors, should be chosen from D_n first.

When solving a problem, LSE gives special attention to the jobs for which the relationship $r_j + d_j = u_j$ is valid. For such jobs that we call fixed, the relationship $s_j = r_j$ holds in any feasible schedule, and for practical reasons it is usually important not to move them in any case. If a fixed job has no predecessors (and this is a typical situation for many classes of actual problems), it is put to its place immediately. If it has predecessors, then they are scheduled as soon as they are put to the set of feasible jobs. Thus it is guaranteed that the fixed jobs will always have their initial starts if only this does not contradict the precedence constraints.

It is possible that the jobs for which their due dates are determined, or their predecessors, are in conflict with fixed jobs and their predecessors in the sense of priority to be scheduled first. This problem can be solved differently. On the one hand, we can equalize these two classes of jobs in their rights to be scheduled first; on the other hand, we may always choose the fixed jobs and their predecessors first from the set of feasible jobs. By default, in LSE priority is given to fixed jobs and their predecessors in this case.

1.7.2 Resource pools support

Work with resource pools is one of the distinctive features of LSE. Recall that a pool is a union of several resources of one type (for example, laptops and desktops, being the resources of different types, can be combined in a pool of all computers). Along with the ordinary resources, a pool can be assigned to jobs in a certain amount. This means that a job can use any resources of a pool in a total

amount equal to that given by the assignment. Thus, a simple resource can be assigned to a job as an independent resource, or as a pool element, also.

Introduction of pools (as well as other extensions) makes the problem, which is already NP-complete and where an exact solution can be found only for small dimensions, even more difficult. So, implementation of exact algorithms for extended statements makes no sense. Any further extensions of the problem statement are considered in LSE only in the context of heuristic algorithms.

So, when fixing a job to which resource pools are assigned, we can use the same greedy algorithm as in the serial method for the base formulation. But in addition to the definition of its start time (s_h), we should also choose which resources exactly will be assigned to this job. An evident heuristics in this case is assignment of the resources that are least of all busy in execution of not yet scheduled jobs. So, for each resource of a pool, we calculate the value $z_r = \sum_{j \in S_n} k_{jr} d_j$ and use first of all the resources for which z_r is minimal. It is also easy, using this extended interpretation of the serial method, to generalize the serial sampling method.

1.7.3 Solution of problems with the jobs of unfixed duration

If we have jobs of unfixed duration, then, when scheduling a job, in addition to its start s_j and the choice of resources from the pool, we should also define the amount of resources used in the assignment which determines duration of the job execution. At the same time, however, nothing prevents us from using the algorithm of scheduling the jobs in a sequence. The following part related to the choice of the amount of resources to be used is added to the heuristics: the job under consideration uses the resources so that its duration is minimized (i.e., it uses all available resources to minimize the lengths of all assignments in the case of sequential use, and to minimize the length of the maximal assignment in the case of parallel use). Resources are chosen from the pool as described above.

2. ENVIRONMENT FOR SCHEDULING PROBLEM RESOLUTION LEDAS SCHEDULER

2.1 Functionality of LEDAS Scheduler

Activity of LEDAS Company in the sphere of development of tools for specification, calculation and optimization of schedules was, from its very begin-

ning, aimed at creating a solution differing in its essence from applications available in the market. The main problems faced by LEDAS team were as follows: creation of the algorithmic base intended to process subdefinite data; the problem of schedule optimization according to arbitrary criteria; specification of requirements to a plan being constructed in the form of arbitrary constraints. These conditions not only show to the best advantage the application created on the basis of methods elaborated for solution of a problem with arbitrary constraints, but make it nearly the only tool able to solve such a general problem.

LEDAS Scheduler 3.0 supports the following entities:

- Jobs
- Resources
 - Unary
 - Consumable
 - Discrete
- Pools of unary resources
- Constraints
 - Precedence, due dates, resources (parameterized constraints included)
 - User-defined (specified in the Scheduler language)

Relationships between jobs and (pools of) resources can be specified by means of assignments: either obligatory or optional. The volume of resource consumption can be defined via assignments.

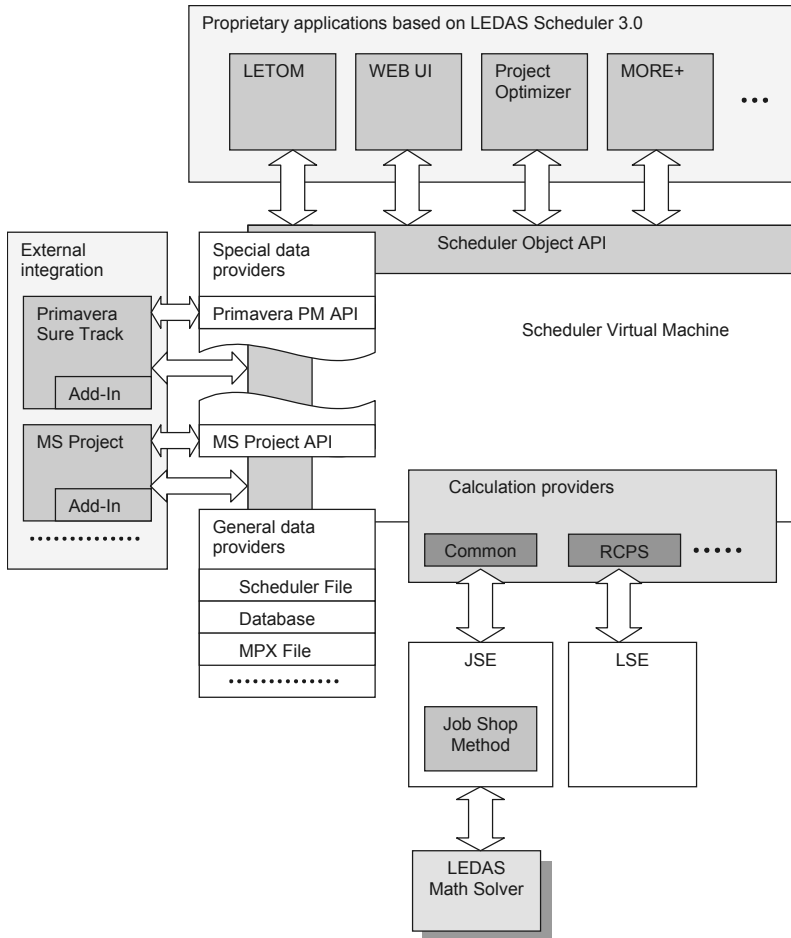
All above-mentioned objects (but not pools and constraints) may contain internal local slots and constraints. The notion of a class defines shared features of the objects' structures. By default, each object is a copy of a certain standard class, but a user can define his/her own classes, thus extending the standard set of slot and constraints. Modification of a class implies modification of all objects of that class.

A planning problem as a whole is represented in LEDAS Scheduler as a special entity — a project. The sets of classes most frequently used can be structured into libraries of classes that can be imported by various projects. The parameterized user-defined constraints can also be specified and included into the library as classes of constraints.

The optimization criterion is specified by a user as a particular goal of the model. Typical goals are minimization of the total duration of a project and minimization of the total resource consumption.

2.2 LEDAS Scheduler architecture

A figure below shows the LEDAS Scheduler architecture from the viewpoint of possible development of the system for solving real-life problems.



The architecture of LEDAS Scheduler has been designed to provide the following:

- Maximal portability to various platforms
- Support of a variety of data formats/sources
- Potential integration of the Scheduler engine into external products
- Extensibility of the Scheduler engine with specialized computational methods

The environment of LEDAS Scheduler 2.5 includes six modules:

- JSVM — a virtual machine of LEDAS Scheduler
- JSEngine — a computational module
- LEDAS Math Solver — an external universal solver
- JSDataProvider — a set of modules for support of various data formats/sources
- JSCalcProvider — a set of modules for integration of specialized computational modules
- JSUI — a graphic user interface.

The version 3.0 additionally has a new computational engine LSE.

JSVM

A virtual machine of LEDAS Scheduler performs all operations on schedule editing. JSVM provides a high-level object-oriented programming interface (API) — a set of abstract C++ classes which includes specialized classes for objects of scheduling problems (IJob, IJSResource, etc.), as well as supplementary classes for work with projects, sessions, providers (IJProject, IJSVM, and others). This API allows us to create several sessions at the same time, to load data from different sources, to modify data, to obtain a solution by using different computational modules, to save the modified model and solutions found, to receive notifications on changes in the project and to combine projects. To save/read data, JSVM calls JSDataProvider, and to recalculate a schedule, it calls JSCalcProvider.

JSDataProvider

This family of modules supports a unified protocol of data exchange with JSVM. Every module of this family provides access to a certain data format/source. At present, three modules have been implemented:

- JSFileProvider saves data in the files of a special LEDAS Scheduler format
- JSODBCProvider exploits ODBC-compatible data sources like MS Access
- JSMPXProvider reads/writes files in MPX format, which allows data exchange with other applications of the class Project Management.

JSCalcProvider

Like JSDataProvider, this is a family of modules supporting the communication protocol of LEDAS Scheduler virtual machine with attached computational modules. For example, the module JSGenericProvider supports communication with the universal computational module JSE (which, in turn, uses LEDAS Math Solver), and the module JSFastProvider serves for connection of LSE, a specialized solver for scheduling problems.

JSE

A universal computational module for solving a wide spectrum of scheduling problems, JSE provides object-oriented programming interface in the form of a set of abstract C++ classes intended to create a model, modify it, and find solutions using different methods and heuristics.

The computational module JSE is integrated into LEDAS Scheduler with the help of a computational provider JSGenericProvider for problems in a general formulation. Since JSE completely supports the whole object model Scheduler (including user-defined constraints), the provider optimally handles the whole model, supports calculation, obtains computed solution, and supports parsing and compilation of user-defined constraints. JSE contains a special efficient method for solving problems of the class JobShop.

LEDAS Math Solver

This is a universal external mathematical solver which has means for modeling a wide spectrum of mathematical problems on subdefinite data with a set of resolution methods. A problem to be solved by LEDAS Math Solver may have an arbitrary structure with various constraints on integer, real, logical and set variables.

JSUI

This is a graphic user interface that provides communication between a user and the system by means of tables, diagrams, etc. User commands are passed to JSVM which, in its turn, provides JSUI with information about objects of a plan.

LSE

The computational module LSE is integrated into Scheduler via the computational provider JSFastProvider. Currently the specialized module LSE can solve a restricted class of problems, so its computational provider verifies whether the problem belongs to this class. If this is the case, the provider states the problem in terms of LSE, passes it to the calculation stage, and receives the solution found.

LSE supports unary and discrete resources, precedence constraints, pools of unary resources, job hierarchy, and due dates. By now, some entities are not fully supported, for example, job duration may be only exact, whereas LEDAS Scheduler environment supports jobs with unfixed duration. Some entities are not supported at all, such as user-defined classes of jobs, resources and constraints, consumable resources, variables and user-defined constraints. They are not required for the statement of a majority of important real-life problems, but, nevertheless, we suppose to support them in next upgraded versions of LSE. There are possibilities supported only by this computational module, for example, interruption and continuation of a solution process, or pools of discrete resources.

2.3 Comparison of LEDAS Scheduler with its competitors

LEDAS Scheduler 2.5 can be compared, on the one hand, with MS Project — a leading application of the project management class, and, on the other hand, with ILOG Scheduler — a special-purpose processor for solving planning problems (see <http://www.ilog.com/products/scheduler>).

2.3.1 LEDAS Scheduler vs. MS Project

MS Project is a convenient tool for a “handmade” construction and maintenance of small and medium project schedules. The system provides a wide spectrum of tools for analysis of a structure and current state of projects: various types of diagrams, reports, sorting, grouping, tables, and many others. A distributed mode of operations on projects is possible — there are tools for multiple network access to data, message delivery, etc. MS Project includes only rudimentary tools for automated calculation of plans and is restricted to detection of data

inconsistency at the editing stage and calculation of feasible schedules that can be far from optimal.

In contrast to MS Project, the LEDAS Scheduler development was mainly aimed at making its computational component as efficient as possible. That's why computational possibilities of MS Project and LEDAS Scheduler are just incomparable — LEDAS Scheduler's tools are much more powerful.

The user interface of LEDAS Scheduler includes a minimal set of tools for project data handling (tables, Gantt charts, PERT, resource diagrams) but it can be extended within JSUI currently available or within a new interface module attached to JVSM.

By now, LEDAS Scheduler does not provide tools for distributed operations on projects, but its elaborate architecture allows them to be inserted without detriment to the system integrity.

2.3.2 LEDAS Scheduler vs. ILOG Scheduler

ILOG Scheduler is a library of C++ classes built on ILOG Solver. These classes are intended to solve planning problems, so they represent entities essential for scheduling domain (resources, jobs, constraints) and provide a set of powerful contemporary algorithms, each being nearly the best known for a corresponding problem domain.

In addition to the algorithms, ILOG Scheduler provides rather flexible schemes for their handling and, taken together, they constitute a set of tools for constructing applications that efficiently solve certain scheduling problems. The library is extensible, i.e., a user can add a new tool (type of constraints or algorithm) by writing a class in C++.

In the Scheduler, similar functions are supported by JSE that consists of a basic solver (LEDAS Math Solver) and an object-oriented library of jobs/resources/constraints/algorithms. In contrast to ILOG Scheduler, LEDAS Scheduler supports the object-oriented model of the problem not only for professional application developers but for end users as well. This means that end users can easily introduce new classes of jobs/resources and define constraints associated with objects of these classes.

JSE allows a high-level statement of problems of a rather wide spectrum (no lesser than that of ILOG Scheduler). Some classes of problems, such as job-shop scheduling, are supported in JSE by specialised resolution algorithms and it is comparable in performance with ILOG Scheduler on these problems. When solving general problems that can be stated in LEDAS Scheduler, the computational power is far from perfect because of the use of a universal method. This problem can be solved by, first, separation of new important subclasses of problems and

implementation of dedicated algorithms for them, and second, integration of external solver, like LSE, via JSCalcProvider

If LEDAS Scheduler and ILOG Scheduler are compared from the viewpoint of constructing special-purpose applications, it can be seen that LEDAS Scheduler has certain advantages. ILOG Scheduler sets aside everything irrelevant to the problem solution. Data editing, storage and exchange with other applications — all these points should be solved by the application developer who uses ILOG Scheduler.

LEDAS Scheduler provides an integrated solution that facilitate building of special-purpose applications:

- A virtual machine of LEDAS Scheduler (JSVM) has an API which supports data editing providing at the same time, as far as possible, an automatic data consistency checking;
- JSDataProvider family supports several data formats/sources for storing of the projects during their design and maintenance. One of these modules, JSMPXProvider, implements data import/export into MPX-files, a standard for data exchange between applications of the class Project Management;
- An appropriate API is provided for advanced application developers who want to create their own JSDataProvider module.

3. APPLICATIONS

We see broad possibilities for the application of the LSE engine and LEDAS Scheduler in general. Scheduling is an essential component of diverse business processes—including project scheduling in the IT sector and equipment utilization scheduling in industrial enterprises, flight scheduling in airlines and scheduling of meetings for a company's employees.

Scheduling is also an important component of product life-cycle management (PLM); therefore the development of LEDAS Scheduler is in line the iPLM strategy announced by LEDAS (see [7] and <http://ledas.com/iplm.php>). Scheduling is needed in almost all phases of a product's life-cycle, from design and prototyping to production and after-sale support. Thus, we believe that an engine for solving scheduling problems can be integrated in a natural manner into full-function PLM systems. LEDAS does not develop systems of this kind on its own and offers cooperation to all companies interested in this.

From the viewpoint of constraint satisfaction and optimization problems, LEDAS Scheduler and the underlying solution represent rather general ap-

proaches stated in general terms. It is a natural desire to specify a problem in a way that matches the specific conditions and business logic of a certain application area.

LEDAS is presently participating in two projects for integration of the LEDAS Scheduler engine into end-user systems—the MORE+ meeting scheduling system and a project scheduling system for the Eclipse platform. Both projects rely on state-of-the art platforms. It is anticipated that they will be able to complete with the existing software of a similar kind. In what follows, we examine the two projects in somewhat more detail.

3.1 MORE+ Meeting Scheduling System

3.1.1 Description of functions of MORE+

The plan to develop a system for automated scheduling of meetings was conceived in LEDAS when the number of concurrent projects became large enough that participation of a specialist in several different projects became a natural and necessary feature of the company's business. Correct organization of work was made more complicated by the fact that some of the projects required outsourcing some highly specialized subtasks to other parties.

This led to the problem of constructing an optimal schedule that satisfies the individual scheduling constraints of every developer in all the projects of the company. The system must provide efficient access capabilities, including the ability to plan and prepare meetings using mobile devices, such as PDAs, mobile and smart phones, and must be able to resolve all of the arising conflicts independently and automatically, with minimum effort required from the participants of the meetings. Ease of use and administration is not mentioned here, since this requirement is present in all projects that do not assume broad unification and cover a large number of specialized applications.

It is anticipated that mobile terminals (such as mobile phones and PDA) will be extensively used to communicate with the system. The scheduling will be done on a server, to which the user will connect via mobile devices (using Wi-Fi or GPRS) by means of a special protocol or via a web-based interface. In the future, plug-ins will be developed for the most common personal scheduling systems, such as Microsoft Outlook or Lotus Notes. A user will be able to initiate a meeting from any terminal, as well as to receive queries on availability and notifications of scheduled meetings.

Such a system can be useful in those companies whose activity requires extensive communications between employees and/or clients and partners. This

may be a dynamic IT company that holds a large number of meetings to discuss ongoing projects, or a large trading company in which meetings are needed so that its employees can exchange their experiences and learn from the others, or a service company in which regular meetings with partners and clients are the very basis of a successful business. Considering that the overall focus of the above-mentioned conditions is rather narrow, compared to the broad algorithmic foundation of LEDAS Scheduler, it is obvious that the Scheduler's solutions must be customized, and the application must be simplified during implementation. In what follows, we will briefly describe the capabilities of MORE+.

MORE+ is intended to automate the coordination of meetings. It will automatically take into account all the events already in the schedule of the participants of a new meeting, as well as their personal restrictions concerning participation in the meeting. The system will not only find a time that is most convenient for the participants, but also reschedule other events, if it is necessary.

Unlike most other systems, MORE+ will combine the individual schedules of the participants, take into account availability of meeting rooms, and output the resulting schedule without requiring the organizer to find a time that is acceptable for all the participants and coordinate this time with all of them.

3.1.2 Mathematical specification of the problem in MORE+

The problem contains the following entities:

1. Set of meetings. Each meeting is a job of a fixed duration, constrained by the use of resources, such as employees, rooms (optionally), equipment, and other resources.
2. Set of individual tasks. Each personal task is a job with a fixed duration and start time that uses exactly one resource, a certain employee.
3. Set of individual constraints. Each constraint in this set specifies that a certain employee cannot participate in a certain meeting at a certain time (for example, because the employee is not ready). It is modeled by a job with a fixed duration and start time that has exactly one non-simultaneity constraint: it cannot be scheduled at the same time with the meeting. This constraint is modeled by introducing an additional resource that cannot be used elsewhere.
4. Set of employees. Each employee is a unary resource that can be used in several jobs, such as meetings or individual tasks.
5. Set of rooms. Each room is a unary resource that can be used in several jobs representing meetings, and every job uses at most one such resource.
6. Set of other resources (equipment, etc.). These may be arbitrary unary or discrete resources.

7. Set of assignments. These are classic constraints indicating which resources are used by which jobs (in the case of a discrete resource, the constraint specifies the quantity of the resource consumed).
8. There are constraints specifying absolute times: for individual tasks and individual restrictions they fix the time of a job, while for meetings they allow changing the time within a certain interval.
9. There is a set of meeting statuses, with each status containing an indication of whether it is set or not, and an initial schedule, containing the times of scheduled meetings and anticipated times of nonscheduled meetings. It is expected that in the process of solution first nonscheduled meetings will be rescheduled, and only then scheduled meetings.

Unlike typical LSE problems, MORE+ problems do not contain precedence constraints (apart from release and due dates), jobs of unspecified duration, non-discrete and consumable resources, and resource pools. Therefore, a MORE+ problem is a special case of an LSE problem in the basic formulation, which can presently be efficiently solved by LSE.

Nevertheless, these problems have some particular features:

1) Large numbers of fixed jobs. However, a special technique that LSE uses to process problems with fixed jobs, allows it to solve these problems in a reasonable way.

2) Emphasis on obtaining a natural solution. This term implies preserving as much as possible the order of jobs and their start times in accordance with preset priorities (scheduled or nonscheduled meeting). In many practical problems producing a natural solution is at least as important as producing an optimal one. At present, a part of the LSE project is concerned with the development of techniques that ensure producing a solution to the scheduling problem that is both (sub-) optimal and natural.

3.2 Project scheduling system for Eclipse platform

Project scheduling systems are widely used at all levels of management. The undoubted leader in this segment is Microsoft Project; millions of copies of this system have been sold, and the sales increase every year. The strong points of this system are its carefully designed and very convenient user interface, a large set of data views for on-screen display and reporting, and integration with applications in the Microsoft Office suite (such as Microsoft Outlook). It has some weaknesses too: its ability to simulate real-life design problems is limited (no resource pools, very limited support for consumable resources, etc.); computa-

tion capabilities for optimal scheduling and risk analysis are almost inexistent; the software runs only under Microsoft Windows—growing acceptance of Linux makes this factor of significant value.

LEDAS and Xored Software are jointly developing a project scheduling system for the Eclipse platform. This system is designed as an improved analog to Microsoft Project. It will run on all the operating systems supported by Eclipse, including Linux. The use of LSE engine ensures better computation capabilities than in the Microsoft product, including optimization of job scheduling for resource constraints. In addition, the system will support a broader class of scheduling problems; the LSE engine will provide scheduling capabilities with pools of alternative resources and with various schemas for the use of consumable resources (for instance, of the consumption of the resources by simple resources).

In the user's view, the system will be similar to Microsoft Project. It will be integrated with development tools for the Eclipse platform, which will make it particularly attractive to the IT companies that use other Eclipse tools.

4. CONCLUSION

The new constraint solver from LEDAS for scheduling problems is currently under development. Its commercial version will be available for licensing during spring or summer of 2005. Approximately in the same time frame the first applications based on it will become available. We invite all the interested companies to participate in the development, marketing, and promotion of LEDAS Scheduler 3.0, as well as to integrate it in their applications.

The new LSE engine for the LEDAS Scheduler project already demonstrates excellent results. On a set of several thousand tests including problems from 15 different classes, the new solver produces suboptimal solutions of good quality, demonstrating exceptionally high performance and scalability. Integration of the new LSE engine into the LEDAS Scheduler environment significantly improved efficiency of the latter.

One important fact about the LEDAS Scheduler project should be emphasized. The new solution expands the family of LEDAS servers: the geometric solvers for CAD tools LGS 2D and LGS 3D (<http://lgs.ledas.com> and <http://lgs3d.ledas.com>), LEDAS Math Solver (<http://ledas.com/solver.php>), a mathematical solver for resolution and optimization of constraint problems, used in LEDAS Scheduler 2.5, and LEDAS Collaborative Solver, a solver for collaborative solution of constraint satisfaction problems in distributed environments

(<http://ledas.com/lcs.php>). In other words, LEDAS Scheduler 3.0 continues the product line related to tools for creation and maintenance of job schedules.

The company is planning to continue the evolution of LSE—both to improve efficiency of solutions and to expand the class of problems that can be solved. At the same time, it will be developing both LEDAS Scheduler and the products based on it. For example, the system for project scheduling will be given tools for analysis of risks, possible scenarios for future development, and costs. The meeting scheduling system will be expanded by new clients for various platforms and applications, and will also include more advanced tools to manage scheduling, such as priorities for meetings and participants. The studies of the market for new LSE-based solutions will help to determine the path of future development of applications and the engine.

REFERENCES

1. Brucker, P. et al: Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112 (1999) 3–41
2. Kolish, R.: Serial and Parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90 (1996) 320–333
3. Le Pape, C.: Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3(2) (1994)
4. Hartmann, S., Drexl, A.: Project scheduling with multiple modes: a comparison of exact algorithms. *Networks*, 32 (1998) 283–297
5. Carruthers, J.A., Buttersby, A.: Advances in critical path methods. *Operational Research Quarterly*, 17 (1966) 359–380
6. Patterson, J.H., Roth, G.W.: Scheduling a project under multiple resource constraints: A zero-one programming approach. *AIIE Transactions*, 8 (1976) 449–455
7. Ushakov, D.: Adding intelligence to software solutions for PLM: New LEDAS strategy. LEDAS preprint, 12 (2004)

A. Ershov, I. Ivanov, V. Kornienko, S. Preis, A. Rasskazov, I. Rykov

**LSE: NEW COMPUTATIONAL ENGINE FOR LEDAS SCHEDULER 3.0
AND PERSPECTIVES OF ITS USAGE**

**Preprint
15**

А. Ершов, И. Иванов, В. Корниенко, С. Прейс, А. Рассказов, И. Рыков

**LSE: НОВОЕ ВЫЧИСЛИТЕЛЬНОЕ ЯДРО
СИСТЕМЫ ПЛАНИРОВАНИЯ LEDAS SCHEDULER 3.0
И ПЕРСПЕКТИВЫ ЕГО ИСПОЛЬЗОВАНИЯ**

**Препринт
15**

Рукопись поступила в редакцию 11.01.2005

Рецензент Д. Ушаков

Редактор О. Дробышевич

Подписано в печать 08.02.2005

Формат бумаги 60 × 84 1/16

Тираж 50 экз.

ЗАО РИЦ «Прайс-курьер»
630090, г. Новосибирск, пр. Акад. Лаврентьева, 6, тел. (383-2) 30-72-02