iSchedule, an Optimisation Toolkit for Complex Scheduling

Anne Liret
British Telecom, Tour Ariane - 5 place de la Pyramide - 92088 La Défense cedex, France, anne.liret@bt.com

David Lesaint, Raphael Dorne, Chris Voudouris
British Telecom, BT Adastral Park - Martlesham Heath - IP5RE Ipswich, United Kingdom, {david.lesaint, Raphael.dorne, chris.voudouris}@bt.com

Most of real-world scheduling problems which include optimisation and feasibility aspects, are NP-hard. To tackle large scale optimization problems, meta-heuristic algorithms are often proposed to quickly obtain near-optimal solutions. However, they have difficulty to find feasible solutions on strongly constrained problems, whereas constraint programming can help as they efficiently handle feasibility problems. Moreover, despite their success, these methods often require complete re-implementation to solve new problems or variations of the same problem. To address these issues, the Framework approach of the intelligent Optimization Toolkit iOpt [5] provides three components for problem modelling, solving and visualizing, based on constraints and meta-heuristic algorithms. The problem representation is well-matched to heuristic search operations, thanks to the use of invariants processing engine. As a consequence it reduces the time necessary to develop efficient solutions. Extensions of iOpt have been developed for specific domains. Such an extension is the iSchedule Toolkit which is dedicated to assist non-expert users in modelling scheduling applications.

This paper outlines iSchedule capabilities and illustrates their use on a particular scheduling problem, the mobile Workforce Scheduling problem, a Vehicle Routing-type problem with time windows, additional tasks and resource constraints, multiple concurrent objectives and subject to disturbances. This problem is of interest to BT in its field service operations.

Keywords: Heuristic Search, Real World Scheduling, Vehicle Routing, Invariant.

1. Introduction

Heuristic Search (HS) techniques are known for their efficiency and effectiveness in solving NP-Hard optimisation problems, though they need to be guided when the problem is strongly constrained and do not guarantee optimal solution. On the contrary constraint programming (CP) methods are known to efficiently solve feasibility problems and can help by removing unfeasible set of task allocations. Thus it is tempting to combine HS and CP to solve constrained optimisation problems. However, there is a need for a software toolkit which supports all the stages of designing and developing a system based on these techniques. For that reason, the Intelligent Systems lab in BT has developed the intelligent Optimisation toolkit iOpt, using object paradigm [5]. iOpt proved its approach on complex domains such as network design, automatic work allocation to field force, resource planning [9], timetabling, and transportation. A major extension of iOpt is the framework iSchedule, a toolkit for developing scheduling systems. iSchedule provides problem modelling and solving facilities based on constraints and heuristic searches.

Following contemporary thinking in software engineering, iOpt [2,3,5,6] allows code reuse to reduce development time for each new problem. Furthermore, application development is additive in the sense that the toolkit is enhanced by each new application, reducing further the effort in developing similar applications to those already included. iOpt is fully written in the Java programming language with all the acknowledged benefits associated with Java. iSchedule benefits from all the same features than iOpt, except they are specialised on scheduling. Especially, specific scheduling applications can be developed without having to review the core part of the system.
This paper outlines the main features of iSchedule (section 2) and describes its use on a particular scheduling problem, of interest to BT in its field service operations: the mobile Workforce Scheduling problem, a Vehicle Routing problem with time windows, additional tasks and resource constraints, multiple concurrent objectives, which is subject to disturbances (section 3). Then the paper concludes on the successful use of the iSchedule framework.

1.1 iOpt: the separation between problem model and algorithms
Based on a framework-oriented architecture, iOpt is designed to support rapid development of e-business applications that require optimisation functions (e.g. planning and scheduling). It is organised into three independent modules: a generic problem modelling framework (PMF) supported by an invariant library (IL), a heuristic search framework (HSF) and visualisation facilities for invariant networks, algorithm configuration and monitoring (Figure 1). This separation between problem model and the algorithm framework allows the user to run different algorithms on different problems at no cost. Further, several tools have been developed which enables the user to monitor the problem model or visualise heuristic search and construction search based algorithms [3]. For more details on iOpt please refer to [6].

![Fig.1. iOpt overview and how to specialise it](image)

Figure 1 shows how to use iOpt/iSchedule to build a dedicated Workforce Scheduling solution. Only a very small part of iSchedule will be extended to meet the customer requirements thus reducing the development time while offering the high-performance algorithms based on iOpt technology.

3. iSchedule: a framework approach for building scheduling systems
iSchedule implements the basic concepts and business rules that are commonly encountered in scheduling problems with unary capacity resources. In addition, iSchedule manages interoperability between heterogeneous systems through XML exchange facilities: load/save a schedule, an algorithm, a scheduling problem specification.

3.1. Problem modelling
Main concepts are reified; a new application is addressed with iSchedule by reusing or overloading these concepts. The implementation can be done in a declarative and functional way thanks to the invariant underlying model.

The problem modelling framework is a high level framework based on an IL where the user can define scheduling problems with unary capacity resources as a Constrained Combinatorial Optimisation Problem (COP) with its decision variables representing resources and task times, a constraint network and an objective function, using the mathematical algebra available inside IL. The utilisation of IL, an originality of iOpt, supports problem modelling, constraint forward
checking, and search methods [7,8]. IL is very useful to quickly test and undo variable assignments [7,8]. Invariants are one-way constraints which, when grouped into a library, form a set of functions that provides a comprehensive mathematical algebra. It allows the definition of relations between variables, and performs incremental updates [1,4] when the value of one or more variables is changed. IL provides a number of built-in data types (Integer, Real, Boolean, String and Object) and operators which are available to the users to state their optimisation problem (figure 2). PMF incorporates solution management facilities by keeping the best solution, current solution or a population of solutions, which can be used by local search or population-based techniques. It can also be instructed to detect constraint violations and stop the evaluation algorithm at an early stage. Concretely, the developer does not have to be concerned with the details of the implementation and can focus his attention on what is specific to his application.

Figure 2 gives an example of invariant-based precedence relation “T₁ starts after T₂ ends”, and sketches how decision variables and heuristic search variables are directly connected to the invariant network. Indeed decision variables in the problem identify a job allocation, i.e. for each task, its resource, the previous activity and the next activity. Search variables identify the positions of a task in the sequences of job allocations which constitute scheduling solutions (see next section); they represent for each sequence the potential positions of a task and the resource index, whose value is mapped to the value of invariants.

The framework provides Java classes which hide the complexity of the decision model from the user, who focuses on problem modelling with entities from his application domain.

- **Activities and resources**: An activity represents any action that a resource can perform. The Scheduling Framework provides two different types of activities: customer activities called tasks and resource breaks. A task corresponds to an activity requested by a customer and that different resources may perform. A resource break is an activity that must be scheduled during the schedule horizon, e.g. lunch, meeting, home time. A resource represents any work unit that can undertake activities in sequence. In the case of Vehicle Routing, a resource is a pair (vehicle, driver) whose combined capacity, capability, and state determine which activities it can or cannot perform. The Scheduling Framework provides fixed and flexible breaks and implements the concept of timeline for capturing time-dependent constraints on allocations.

- **Timelines**: Various resource attributes such as vehicle capacity affect the allocation of tasks to resources. Resource attributes may also evolve over time due to the execution of tasks. For instance, the load of a vehicle is constrained by its capacity hence preventing certain allocations to be made. The load is also modified whenever goods are delivered or picked-up at customer premises. Timelines simply capture the evolution of attribute values over time. The Scheduling Framework provides three types of timelines: capability timelines, capacity timelines, and state timelines.

- **Tasks**: A task is primarily defined by a duration, which may depend on the resource assigned. A task also features a start time and an end time which are connected by the usual relationship \( \text{start time} + \text{duration} = \text{end time} \), which is redefined in the case the option for splitting tasks over
The framework allows disabling tasks or resources, which will be then ignored in the further solution search runs.

- **Schedule**: A schedule is defined by a sequence of activities for each resource and a list of unallocated tasks together with global consistency and cost information on the schedule. The underlying computational model is the network of invariants representing the various constraints and costs that are built-in or user-defined.

- **Constraints**: The framework allows the statement of precedence constraints, resource constraint (same resource or not), time windows on start times or end times, resource compatibility with tasks.

- **Parallel relations**: They enforce several tasks to start at the same date simultaneously. A delay may be allowed which means there may be a maximum time gap between parallel tasks. In the case where resources are one-capacity, tasks are performed on different resources, which implies the checking of resource availability too.

- **Task costs**: They depend on the duration of the allocation, due dates, and the resource utilised for the allocation. For each task, a cost is incurred for not allocating the task in the schedule (unallocated cost), for letting a set-up period, for delaying the start of the task (lateness cost per unit of time), or for having a resource process the task (service time cost per unit of time).

- **Resource costs**: They rely on wages, overtime or travelling hours. For each resource, a cost is incurred for using the resource, for using the resource in overtime -whatever the task being processed (overtime cost), for using the resource per unit of time on any service work. The set-up time cost, for using the resource on any set-up period, can model a travelling cost too.

- **Service and set-up model**: They contain sub-models for the user to define their own cost, compatibility relation, or duration estimation of durations. They enable modelling resource setup/travel times and task duration. These are sub-models that can be easily linked to external systems, such as geographic information systems in the case of travel times, to offer realistic estimates. The cost sub-model is dedicated to contain as many terms as required to capture the particular business problem.

### 3.2 Scheduling Algorithms

Algorithms in iSchedule make use of heuristic search (through HSF), coupled to constraint satisfaction, but may be extended so as they integrate other types of algorithm, as graph theory, integer programming. Initially, the Heuristic Search Framework (HSF) was created to be a generic framework for the family of optimisation techniques known as Heuristic Search. It covers single solution methods such as Local Search, population-based methods such as Genetic Algorithms as well as hybrids combining one or more different algorithms. In HSF, the functionality of common HS algorithms is broken down into components (i.e. algorithmic parts) for which component categories are defined [2,5].

- **Search Component** represents the basic concept that could be encountered in an heuristic search, for example, a Neighbourhood in a Local Search, the Aspiration Criterion. A complete algorithm is a valid tree of search components (A valid tree is a tree of search components that can be executed).

- **Heuristic Solution** is the solution representation of an optimisation problem manipulated inside HSF, in the case of iSchedule, a set of sequences.

- **Heuristic Problem** is only an interface between an optimisation problem model implemented using PMF and HSF.

Per se, the Scheduling Framework does not make any scheduling decision. This is the role of external processes such as heuristic search algorithms. The framework simply provides the necessary interface to support schedule modifications: An insertion operation, which inserts a task before a specified activity in the schedule (tour construction); a swap operation, which swaps a task with a specified task in the schedule (tour improvement). These two operations suffice to
implement any complex move operator/neighborhood. Of course, search algorithms require more than simple schedule modification operations, e.g. checking consistency, costing schedules, providing access to feasibility and cost information, saving the best solution encountered, resetting/swapping solutions, etc. These requirements are not specific to the scheduling domain but arise in all optimisation problems, which is why they are provided at the level of iOpt. For example, Figure 4 shows a scheduling-specific Fast Local Search algorithm where some algorithmic parts come from iOpt (red colour), other from iSchedule (green colour) and parts are newly created by the user (user-defined Task Selector for instance, in blue) to express a specific way to select the subset of tasks to be considered for insertion by the neighbourhood component.

Fig.4. Example of a tree of algorithms parts for a Fast Local Search algorithm based on task insert, swap and segment exchange neighbourhoods.1

Among the parts readily available in iSchedule, we find the most famous components from local search operators such as segment neighbourhoods (2-opt, 2-opt*, OR-opt, Cross-exchange) and Critical Block based neighbourhoods, or initial schedule generation ("Longest Job First Generation", "Hardest Job First Generation", etc.) to population-based operators such as crossovers ("Uniform Resource Crossover", "Uniform Task Crossover" and "Route Crossover").

3. Application to real-world field force scheduling

iOpt and iSchedule have demonstrated their ability to provide quick prototyping of customised systems and solutions. Experience gained has been utilised to enhance the system, reducing further the effort in developing similar applications to those already included. Among these applications is the NGDS system (Next Generation Dynamic Scheduler), scheduling the routes of BT field engineers. With more than 150,000 tasks to be managed by 30,000 techs everyday, the BT workforce scheduling problem is a large scale problem [10]. Moreover the goal is quite challenging, covering service provision to business and residential customers, network maintenance, and fault repair, including the allocation of work, the execution and the tracking of jobs.

1 FLS searches sub-neighborhoods of a given neighborhood using a given neighborhood search, deactivating those sub-neighborhoods which have no promising moves for future iterations or until they are re-activated as the result of a performed move. The search receives an input Heuristic Solution that is modified and returned as its current solution. A Schedule FLS support is a class describing the way the neighborhood search used by FLS works. A neighborhood search evaluates a set of moves provided by one neighborhood in order to select the next move, or moves to apply its current solution. Best Improvement goes through the list of moves provided by the neighborhood and performs the one that makes the best improvement to the solution.
The mobile workforce scheduling problem can be stated as a Vehicle Routing problem with time windows, additional tasks and resource constraints, and multiple concurrent objectives. The problem consists of mobile skilled resources, lunch breaks and breaks representing the night period (called home breaks), located tasks with time windows on start and/or end dates, long jobs which are likely to be split over breaks to be allocated, and parallel jobs. The objective of the problem is multi criteria: minimize the lateness and the number of unallocated tasks, minimize the travel, balance the resource usage so that at least each resource has at least one job per day, and minimize overtime. In short, it is about assigning the right job to the right resource at the right time to the right route with the right equipment.

The underlying computational problem is composed of a feasibility problem where the proposed allocations meet the constraint requirements so that no inconsistency occurs during execution, and an optimisation problem whose solutions agree with objectives. Multiple objectives become conflicting optimisation criteria due to schedule horizon restrictions, consequently leading to a problem where identifying a good quality solution (the greatest number of criteria satisfied) is time-consuming. To handle the complexity of multiple objectives, NGDS explicitly makes objective terms available to the user, so as they can adjust cost expression, before the schedule search engine performs global optimisation. In addition NGDS complies with a variety of requirements that real-life scheduling domains must comply with: skill matching, routing, due dates, staff breaks, working time regulation constraints, task splitting on the fly. Built-in costs cover standard scheduling objectives and can be specialised to fit into certain requirements such as Quality of Service, travel and overtime penalties.

Typical outputs of NGDS include a Gantt Chart, table of task with information about their allocation, search progress view and statistics report (Figure 5). The tool offers a lightweight risk management facility as it helps making decision before task become failed business targets. Indeed it displays task allocation in different colours depending on task type, priority (importance score related to business target). Thus work managers can deduce task allocations that are candidate to jeopardize such as tasks about to fail (scheduled but too late or disruptions cause the task to be started or ended too late with respect to its due date or task not dispatched whereas its target is close to current time e.g. within the next hour). Also managers would be likely to distinguish an event that puts in jeopardy all the jobs in a tour from an event that just endangers the last job of the tour.
Fig. 5. Typical outputs of NGDS. Task allocated in advance are displayed in blue, in green when scheduled on time of the expected day, in orange if there is up to one day late, and in pink tasks which would be allocated at the earliest because their target is already failed at the schedule start date.

Moreover the mobile workforce scheduling problem is subject to disturbances. In this particular context, to handle dynamic scheduling, a backend optimisation engine (predictive scheduling) is combined with online allocation capability (reactive scheduling). Estimated schedule is periodically (re)computed (rescheduling phase); the task allocation and input data are constantly changing while the schedule is executed during the day. To reduce the complexity of rescheduling we reuse past work, using the existing schedule as a starting point for the next scheduling. It suffices to identify and throw away invalid portions of the existing tours to build new tours. Also we extended iSchedule with capabilities which distinguish three sections in the schedule, the past, current and future schedule, depending on a current time of execution and an offset. The separation of the three schedules allows enforcing different consistency constraints on the schedule (for instance tasks which are near to be dispatched cannot be moved) and applying different solution search methods on each part.

Thank to iSchedule, the scheduling engine has been very quickly designed and developed (2 months with 2 persons). The remaining thing users would have to do was the algorithm and objectives fine-tuning (via graphical tool). We made intra-comparisons between results obtained with different algorithms configuration (figure 6).

4. Results

This section talks about results obtained when testing NGDS engine on a particular instance. The real-world daily region-localised problem deals with about 1000 jobs, up to 200 resources, and a scheduling horizon between 1 and 2 weeks. Jobs have start target dates ("startby" task), time windows (appointment) or completion target dates ("completeby" task).

Approaches used combine: 1) constructive search, that is first Schedule reuse, then parallel job insertion, long job insertion, then unallocated jobs insertion which can be random, task-importance ordered, technician tour-quality based. 2) Hill-Climber or Fast Local Search based on insert/swap/segment relocate moves, a GLS component to escape local optimum (some of the tasks are never visited due to a local good value found for travel cost), and best improvement neighbourhood search. It has been noticed that constructive search takes at least half of the running time (up to three minutes) and it is worthwhile treating long and parallel jobs beforehand as they are strongly constrained.

![Table of statistics](image)

Fig. 6. Table of statistics

The statistics table above shows number of scheduled tasks, average travel time per technician in minutes, and the number of jobs scheduled per day in average. These values are given on the right
column for the schedule in day 0 (first day of scheduling) and as an average over all days in the left column. For the run of FLSGLS_TO, the second table shows the percentage of scheduled jobs per category of jobs; where jobs are organised into categories depending on their due date and type. For a problem of 732 assignable jobs and 135 technicians, instance inspired from real workload of a Tuesday in June 2002. The run lasts 180 seconds. For this instance, FSLGLS evaluated 13870 moves in 85.35 seconds, in a network of 138927 invariants, and with a total of 248316 propagation sessions performed.

All algorithms allocate a major part of the jobs, though FLS does better on that aspect, while keeping an affordable total travel cost. Hill-Climber ends up with a higher travel but it is partially due to the fact it accepts allocating tasks that are very far, though HC needs to have the travel cost weight higher than for the other algorithms, so as to maintain bounded travel. FLSGLS gives good compromised solution w.r.t. allocation and travel. Especially when looking at the statistics per task category (figure 6), we can see that NGDS achieves most of the time between 95% and 100% of scheduled tasks. Constructive search TB is focusing on the technician tour whereas TO inserts tasks according to the business importance (the highest before). RO performs task insertions where tasks are picked-up in a random order. The initial schedule computed by the TO constructive search is improved by the fast local search, keeping high priority (HP) tasks scheduled, as the percentages for HP tasks categories is high.

5. Conclusion

In society when the need of saving and rapid Return On Investment is increasing, object-oriented Framework approach looks promising to produce and maintain real-world scheduling systems. As an evidence, successful application of iSchedule and integration to client tools (mobility devices for job execution, resource planning, people management), as well as interoperability with web-based services. The iOpt/ iSchedule package constitutes now a twofold offer in optimisation and scheduling: firstly a strong technical capability in modelling, solving and visualising complex problems inherited from iOpt, and secondly a high-level framework that allows rapid development of scheduling applications.

References


MISTA 2007