When the Rubber Meets the Road: Bio-inspired Field Service Scheduling in the Real World

Israel Beniaminy[1], Dovi Yellin[2], Uzi Zahavi[3], and Marko Žerdin[4]

[1] ClickSoftware Ltd. Israel
israel.beniaminy@clicksoftware.com
[2] ClickSoftware Ltd. Israel
dovi.yellin@clicksoftware.com
[3] Bar-Ilan University Israel
zahaviu@cs.biu.ac.il
[4] Zany Ants Ltd. UK
marko.zerdin@zanyants.com

Summary. We discuss a class of large-scale real-world field service optimization problems which may be described as generalizations of the Vehicle Routing Problem with Time Windows (VRPTW). We describe our experience in the real-world issues concerned with describing and solving instances of such problems, and adapting the solution to the needs of service organizations using a ”universal framework” for bringing together various problem representations and experimenting with different algorithms. Implementations and results of several bio-inspired approaches are discussed: Genetic Algorithm (GA), Ant Colony Optimization (ACO), and a hybrid of ACO with GRASP (Greedy Randomized Adaptive Search Procedure). We conclude by discussing generation of ”human-friendly” solutions, through introduction of local considerations into the global optimization process.

1 Introduction

We discuss a class of large-scale real-world field service optimization problems which may be described as a variant of the Vehicle Routing Problem with Time Windows (VRPTW) with more elaborate instance description, a variety of additional constraints, and a variety of components for the value function which we seek to maximize.

In this section, we describe the problem and its business relevance, its variants and complexities, and lay a foundation for a subsequent discussion of practical methods for solving the problem. In section 2 we describe a universal architecture capable of accommodating a large variety of problem variants and providing a framework that can be utilized by various search algorithms. In sections 3 - 5 we address a few selected bio-inspired algorithms that we tested and present their corresponding results. In the conclusion of this chapter, we introduce some additional ideas and present conclusions.

1.1 Background and Problem Classification

Field service organizations exist in a dynamic, ever-changing world which combines long-range planning with emergency responses. While this could be said to apply to most types of businesses, there are two features of field service that make it particularly challenging. First, the main "raw material" used in "producing" field service are work hours which must be applied at the right time – when the service is provided. Unlike tangible raw materials, and unlike work hours used to produce tangibles that can be stored, a supply of work hours cannot be stored until a time when there is a large demand for it. Field service managers describe this fact as "work hours have zero shelf life", or even more briefly as "use it or lose it". The second characteristic of field service is that resources and demands all have physical locations (e.g. location of the service engineer, location where the task needs to be performed). This brings travel times to the forefront of any attempt at optimization.

Previous research on similar problems is reported in [6, 9, 10, 13, 15], and studies of such problems in commercial settings in [16]. Our own experience with these problems comes from our work with a commercial software vendor in the field service optimization space, which gives us the opportunity to discover the real-world issues concerning describing a problem, solving instances of the problem, and adapting the solution to the needs of service organizations.

Generally speaking, field service optimization requires finding a matching of resources (employees, tools, vehicles etc.), demands (jobs, tasks, outages, emergencies) and times. As in all optimization problems, such matchings need to be valid – they need to meet specified constraints; and they need to produce a "good quality" solution – that is, achieve a high value for the goal function. Service organizations differ in their needs, and therefore they differ in the required representation for resources, demands, constraints and goal functions.

These four components of problem representation are not independent. In our experience, they are dictated by four categories of business drivers:

1. *Customer Satisfaction:* The matching of resources, demands and times should meet customer expectations regarding the performance of service: sending the right workers (e.g. with the right skills) to the right location, with the right tools and parts, at the right time (e.g. promised appointment window). Failing to do this has at least two negative consequences: if customers are dissatisfied with the service, it can lead to reduction of business; and tasks that were not performed properly (or at all) may have to be repeated, adding to the workload and to the costs.
2. *Employee Satisfaction:* The service must conform to the employment policies: it has to be limited to working hours and a reasonable amount of overtime, respect times when employees are unavailable, and allow for lunch breaks during the day. Failing to meet these constraints may lead to losing the organization's most important assets – its workforce – as well as involving it in disputes with labor unions and employee representatives.
3. *Finance:* The service must be performed in a cost-effective manner, considering factors such as the cost of travel, the cost of using different types of workers, the cost of overtime etc. Obviously, workforce utilization is a major component of cost; both "idle time" and excessive travel represent work time that must be paid for but does not contribute to meeting the demand for service and does not bring any revenue.
4. *Regulations:* In many cases, there are regulations governing the service process - e.g. a statutory limit to response time for a call, depending on its severity. Failing

to meet regulatory requirements may lead to substantial fines or even to a loss of operating licenses.

Since different field service organizations have different sets of policies and goals, this general description gives rise to a large and varied class of problems. In the following section, we present a formal way of classifying and describing constraints that may be included in each instance, depending on whether the constraint in question controls the validity of scheduling a demand, assigning to a specific resource, scheduling at a specific time, or any combination thereof.

The same classification may be used for components of the goal function. However, in the real world the distinction between constraints and goals is not always clear-cut. For example, time windows are typically considered to be a constraint, but many service organizations feel that it is better to arrive late than to fail to arrive at all. Furthermore, "being late" is itself a fuzzy concept – e.g. being late by ten minutes may be acceptable, missing by thirty minutes may cause some customer dissatisfaction, while keeping the customer waiting for an extra hour may cause the customer to switch to a different service organization. Another negative consequence could be a wasted trip, because the customer will not wait for the technician. Real-life considerations of this sort sometimes call into question the classification of problem terms into constraints and goals.

1.2 Problem Attributes

Taking VRPTW as the baseline problem description, we find that even the simplest types of field service problems have at least one additional characteristic that makes them fundamentally different from VRPTW. Each such characteristic typically applies to two or more of the parts of problem specification mentioned above: resources, demand, time, constraints and components of the goal function.

Here, we systematically classify some of the more common characteristics of this large variety of problems. The letters `D`, `R`, `T` stand for demand, resource and time, respectively.

- Resource characteristics
  - Resources usually have their own *individuality* which is expressed by attributes such as skills, certifications, efficiency (e.g. some resources consistently finish 10 home-base location, and "calendar" (times at which the resource is available).
  - In light of this individuality, resource capacity is usually limited in a much more rigid way than with the classical VRPTW. Just throwing "another vehicle" at the problem, a common method of achieving feasibility with VRPTW, is usually out of the question.
- Demand characteristics
  - Demand attributes typically include location, allowed time window, expected duration, and required skills. Other attributes that we often see include the revenue generated by performing the task (or a cost of not performing it), priority, and dependency of demand duration on other parts of the solution (e.g. if a task is scheduled immediately after another demand performed by the same resource at the same location, its duration may be reduced).

  - Demands without a uniquely-specified location: Some demands may be fulfilled remotely, from wherever the resource happens to be. Other demands may be fulfilled in one of a number of locations, so that the schedule needs to select one of the allowable locations when assigning the demand to a specific resource and time.
  - Number of required resources: Some demands require more than one resource, specifying the required attributes for each resource.
  - Requirement for parts: Some demands need parts – consumables, spare parts or tools. Vehicles assigned to service engineers may have their own "in-van" inventories, in which case vehicles which already have the required parts are a better choice. When the required parts are not immediately available, a separate "pick-up" sub-task needs to be added for getting the part before driving to the service site.
- `DR` interactions - affecting validity or value of assigning `D` to `R`
  - Matching resource and demand attributes: Demands almost invariably require certain skills and only resources that possess those skills can be assigned to them. Another example is resource "work zones", in which case each matching depends on geographical relationship between demand's location and resource's home-base.
  - Matching specific resources:
    Some demands specify a "required engineer" (who is the only one allowed to perform the specific task) or a "preferred engineer" (so that assigning this engineer raises the goal function's value).
- `RT` interactions - affecting validity or value of scheduling `R` at `T`
  - Resource availability: The resource's calendar specifies when the resource is available to be assigned to demands.
  - Resource time-dependent costs: Resource calendars often specify varying costs depending on time of day, as in overtime work.
- Interactions between assignments
  - Interdependence between demands: Some demands are actually parts of a larger job, and in these cases it is common to see that the decisions made for one demand affect the feasible options of another. Such dependencies usually apply either to timing (e.g. when demand B must not start before demand A was completed) or to resources (e.g. when the resource assigned to demand B must be the same as the one assigned to demand A).
  - Non-local constraints: Often, there are constraints on some aspects of the solution that can't be expressed locally (that is, for one specific assignment or for a single arc between consecutive assignments), but instead depend on the solution as a whole or on a non-trivial part of the solution. For example, even when each resource is limited to a maximum of two hours of overtime per day, there may be a fixed maximum for the amount of overtime any single resource can do throughout a work week (possibly due to union rules), or a maximum on the total amount of overtime in a region (due to budget limitations).

The above list, of course, is just a small sample of what we see in the industry. We could give many additional common (and unique) examples for constraints and value-function components, some of which our formalization places in additional categories such as DT and DRT.

From a different point of view, we further classify field service scheduling problems into on-line and off-line problems. The scope of this chapter does not allow a detailed discussion of on-line field service optimization, but we note that the on-line variants are very important in real-world applications. One of the goals of on-line variants is to take a given solution, change the problem instance (e.g. by modeling the fact that a demand has extended beyond its expected duration), and find a solution for the new instance that meets the constraints and optimization goals while minimizing the number of changes[1] compared to the original solution. Another important goal of on-line optimization is to provide a prompt response when necessary – the customer won't happily stay on the phone for a long time while waiting for a list of appointments to choose from.

1.3 Problem Complexity

Even small service organizations regularly encounter large problems. For example, take a 10-engineer service organization which performs, on the average, five tasks per engineer per day. If this organization wishes to plan a week ahead (five work days), there are 250 demands in the problem. Our experience extends to workforces with tens of thousands of resources and hundreds of thousands of demands. Such problems are, to some extent, separable, since resources located hundreds of kilometers apart usually won't be considered for the same demand, but factors such as overlapping regions make strict separation difficult. Furthermore, even geographically-localized subproblems can easily reach 30,000 demands.

The intractability of VRPTW for any but the smallest problem instances is well-known. It is worthwhile mentioning that intractability remains even when travel times are all zero ([12, 13]), and even when further requiring all demands to select from a non-overlapping set of time windows (leading to variants of General Assignment Problem - see [5]). Recently, it was found that even when the time window for each task is only slightly longer than its duration, and when all travel times are zero, the problem is still (weakly) NP-hard [1]. This observation is relevant to problem instances we encountered where many travel times are indeed zero (work within facilities) and when most time windows are set by fixed appointment slots (e.g. morning or afternoon appointments).

2 Solution Architecture

2.1 "Universal" Model for Representing Scheduling Problems

The need to support varying business policies of different types of field service organizations has led us to define the **W-6** model, whose name stands for "Who does What, Where, When, With what and for Whom". This is a generic model allowing us to describe the full richness of real-life field service scheduling problems. To make this

[1] The requirement for considering the number of changes as one of the optimization criteria has several reasons, one of which is the propagation of each change to other enterprise software and to the human resources being scheduled.

concrete, we note that the simplest solution description is a set of triplets $\langle D, R, T \rangle$, each triplet specifying that demand D (What) is assigned to resource R (Who) at time T (When). More complex descriptions add additional dimensions, leading to or beyond the six dimensions implied by the name W-6. The W-6 terminology can be viewed as a language in which words are demands, resources and other problem building blocks, the grammar is composed of scheduling policies and logic, and the resulting sentences are the assignments.

2.2 "Universal" Field Service Scheduling Framework

With the **W-6** model as our guide, we defined a framework for representing and solving field service scheduling problems. The design goals for this framework were:

- Accommodate steps, roles and concepts that are a part of most common field service scheduling problems.
- Allow the framework's building blocks to be replaced or extended by customization, so that the framework would support the features that were not part of the original design or were not even anticipated at design time.
- Support a wide variety of optimization problems encountered in real-life field service situations: In order to achieve that, we need to break up the relevant logic into small modules, each responsible for implementing a certain business constraint or goal. These modules are then used together, describing a specific field service problem like adjectives describe a noun.
- Allow the decoupling of complexities associated with creating valid, constraint-satisfying schedules from algorithms that can be used for optimizing those schedules: In order to achieve that, we need to expose the interfaces that allow the schedule-building logic (and its parts) to be used as building blocks for the assembly of working algorithms.

The functionality of the framework needs to support the goals of schedule optimization. We define these goals as: *To create and maintain a collection of valid, constraint-satisfying assignments (a schedule) while providing the services needed to optimize the quality of this schedule.* Figure 1 gives the overall structure of the framework, where a set of logic services encapsulates access to W-6 objects and to computations (such as constraint propagation) performed on these objects. The framework allows additional components (such as custom constraints) to be "plugged-in" and enables any such components to seamlessly participate in the performance of the required services.

From the point of view of scheduling algorithms, the framework is represented by the logic services that it provides. In the remainder of this section we describe these services.

Creating and maintaining W-6 objects

The lowest level on figure 1 – the object server – abstracts the data-manipulation of W-6 objects and frees the higher layers to concentrate on the logic of building

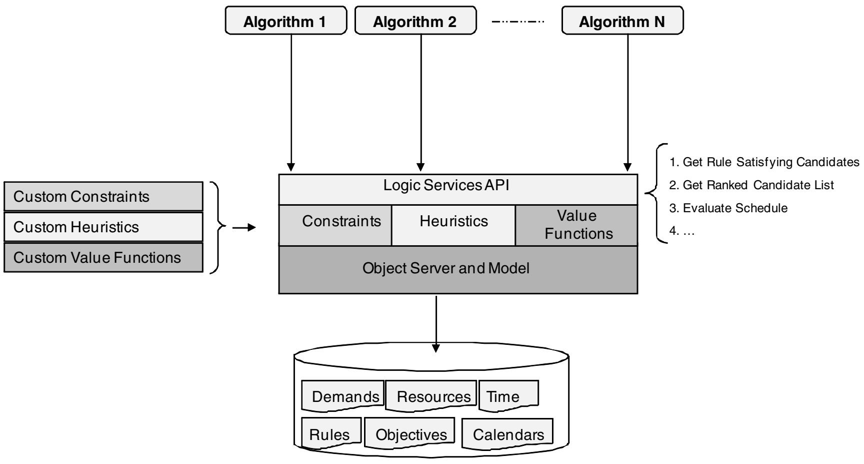

Fig. 1. A schematic view of the universal scheduling framework

constraint-satisfying schedules as well as optimizing these schedules. A few features of the object server deserve to be specifically mentioned:

- Creating, setting property values of, storing, querying and destroying any W-6 object.
- Maintaining certain structural invariants of W-6 objects, e.g. the difference between assignment's start and finish always needs to equal its duration.
- Triggering events before or after the methods are executed, in order to allow the implementation of custom business rules, dynamic property calculation and communication with other information systems.

Constraint-propagation engine

Constraints, both built-in and externally defined, are implemented in a constraint-propagation system tailored specifically for field service optimization problems. This system also includes propagation of temporal constraints. Constraint propagation generally works by successively eliminating possibilities that have become invalidated by the scheduling decisions that were made (usually by the currently active algorithm). For example, if a demand was assigned to a resource, other demands that can't fit into the remaining availability will be eliminated as candidates for that resource.

By providing a generic constraint-propagation engine and by allowing custom constraints to be incorporated, the framework essentially allows any field service problem to be modeled, no matter how exotic its constraints. Furthermore, the algorithms described in this chapter are generic and independent of the specifics of the constraints for each problem instance. Without this decoupling, separate algorithms would need to be developed for different combinations of problem characteristics, leading to an exponential explosion in number of required algorithms.

Ranking and heuristics

Once constraint-satisfying assignment candidates have been produced, they become subject to the scheduling decisions by the algorithm. Most algorithms include basic steps in which they need to construct or modify a solution by creating a new assignment and binding some or all of its slots to concrete values – e.g. selecting a specific resource for the `R` part of an assignment. Due to constraint propagation, the allowable values for selection depend on static problem data as well as on the solution to which the new assignment is being added. These selections are almost invariably guided by heuristics that are used to sort, weigh or otherwise rank the candidates. The framework provides a number of built-in heuristics that can be combined to model the scheduling decisions employed by dispatchers. It also allows the introduction of custom heuristics. These capabilities allow the algorithms deployed on top of the framework to transparently and consistently (across the algorithm stages) order the candidates in any order they need for their optimal performance. In the simplest case, a greedy schedule building algorithm is implemented by always selecting the highest ranked candidate, and the framework will automatically do the rest. More sophisticated algorithms have the same heuristics available to them, leading to consistent and fast design and testing of algorithmic ideas over any type of problem which may be modeled in W-6.

Evaluating the schedule

Once the schedule has been built, the algorithm usually has to calculate its value in order to know how to proceed. The framework allows the value to be composed of any number of business measures defined per assignment, assignment pair or globally. Again, if built-in measures do not provide sufficient expressiveness, the framework allows the incorporation of custom measures of any complexity. The introductory section describes a number of components of schedule value which are often encountered in practice.

Evaluation also provides information about specific parts of the problem which may indicate opportunities for improvements, e.g. demands which aren't served by the current solution or unused resource availability.

3 Genetic Algorithm

In this section we describe how a genetic algorithm (GA) was developed and applied on top of the framework described above. The framework enabled a drastic reduction in development effort, and the resulting GA was more flexible and able to efficiently handle large, complex, real-life problems.

3.1 Implementation

The general structure of genetic algorithms is well known (see [7], [2]), so we won't repeat it here. Instead, we will devote this space to implementation details specific to the field service optimization problems.

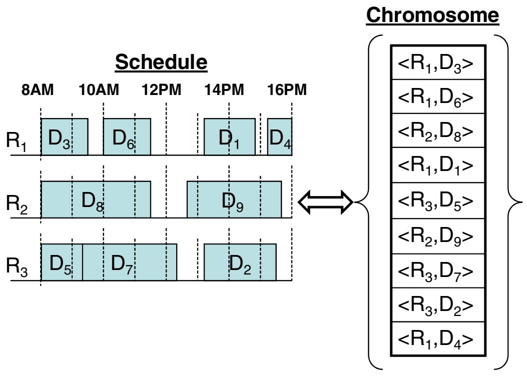

Fig. 2. Relationship of the chromosome structure to the schedule (the gaps between demands correspond to travel times and lunch breaks)

Chromosome structure

Chromosomes are the lifeblood of genetic algorithms. They don't just represent solutions, they also need to have a structure that allows evolutionary mechanisms – mutation, recombination, selection – to work effectively.

Choosing a chromosome representation which is equivalent to the solution representation (e.g. triplets specifying which resource performs which demand at what time) leads to an inefficient algorithm because of the low probability that a mutation or crossover will "squeeze" demands so that they are separated by the exact time required to travel between them. Instead, we chose to represent a chromosome as a "recipe" for generating a solution. This recipe takes the form of an ordered set of $\langle R_i, D_j \rangle$ pairs, specifying that resource i is assigned to demand j. We refer to these pairs as genes. The number of genes in a chromosome is equal to the number of demands in the problem instance. All genetic operators maintain the invariant that each demand appears exactly once in the chromosome. During the evaluation of the fitness function, a solution-generation engine scans the genes in the specified order, and attempts to schedule each demand at the earliest possible time after the other demands already assigned to that resource. This is done be relying on the framework, which considers travel time from this resource's previous assignment as well as any other constraints defined for this problem. This chromosome structure, and the process of generating a schedule from it, is shown in Figure 2.

The order of pairs in the chromosome is significant, since demands are appended only to the end of their resource's current route. For this reason, and for reasons related to the random nature of mutation and crossover operators, not all pairs correspond to permissible assignments. The scheduling framework detects this situation and reports it to the solution-generation engine. When such a situation occurs, the engine triggers a method attempting to locate another candidate allocation from the list supplied by the framework, and use it instead. This approach can be classified as belonging to the family of "repair strategy methods" (see [2, 11]). Note that in many problem instances it is difficult or impossible to find a schedule in which all demands are included. Therefore, it is common to find genes towards the end of the chromosome which can't be translated into valid assignments even after applying the repair strategy.

It is tempting to "fix" the chromosome by changing the gene which generated the infeasible assignment so that it reflects the assignment found by the repair method. However, our tests show this Lamarckian mechanism to exhibit lower performance, and we don't use it in the GA. This finding is in line with reports such as [17], where Lamarckian inheritance was found to have a higher tendency to converge to a local optimum, while utilizing the Baldwin effect had a better chance of finding the global optimum. We note that the repair method may be viewed as a limited Baldwin-inspired search, activated only when the genotype fails to create a valid phenotype.

The chosen chromosome representation has some redundancy. It is clear that many permutations of genes are equivalent – causing the engine to generate the same schedule – as long as the permutations preserve the ordering of the genes having the same resource. [2] We could remove this redundancy by always sorting the genes in some predefined order of resources without changing the schedule which the engine builds from each chromosome. We allow the redundancy, even though it increases the size of the search space, since this representation increases the probability for the operators that we use to find one-step moves which increase the solution quality.

Initial population

The creation of schedules that constitute the initial population is based on the framework's capabilities for solution construction. In order to generate a diverse initial population, the level of variation in ordering and selection of demands and resources should be high. We therefore employ a degree of randomization while still aiming at selecting heuristically good candidates. We have found that a good selection of heuristics in the construction of the initial population, combined with a suitable randomized selection of the candidate assignments as ranked by these heuristics, is important for the success of the GA as a method of finding high-quality solutions. In fact, we also developed a GA that searches for good heuristics, but space limitations preclude description of that GA.

Successive generations

Given a population, we use the following method to create the next generation:

1. *Segmentation by operators:* We divide the population into fixed-size segments, each of which is subject to a different operator. One of the segments is the "elite" segment, preserving the top few solutions from the previous generation. While this can lead into a premature "lock-in" of the population into a local optimum, this method has performed better in our studies than methods which allow the maximum quality to decline between generations.
   Each of the other segments is dedicated to one operator out of a variety of mutation and crossover (recombination) operators. We have found that the specific mix of

[2] For simplicity, we ignore the repair method in this paragraph.

operators is less important than the fact that they perform a wide variety of actions in order to more efficiently explore the search space. Some of these operators are described below.

2. *Choice of parents:* Each operator needs to select one or two parent chromosomes from the preceding generation. The operator acts on these parents to create a new chromosome in the current generation. After some experimentation, we decided on uniform probability for selecting any parent from the previous generation. In our experiments this performed better than other methods in the literature, such as competition or biasing the probability of selection towards higher-value solutions.
3. *Applying the operators:* Finally, we apply the relevant operators to the selected parents, which results in the new generation of chromosomes.

Crossover

There are numerous approaches to applying crossover and mutation to various scheduling problems in the literature (e.g. [14]). We find the most appealing choices to be variations of the "edge recombination" and the "order-based" operators (see chapters 21 and 22 in [2]). Both can be implemented for the field service scheduling problem, though they require substantial modification.

Our implementation has focused on order-based combinations, whose purpose is to create child chromosomes which preserve some of the ordering of demands in both parents. In our case, this affects the order in which demands will be sent into the solution-generation engine when a schedule is constructed from a chromosome. The re-ordering may change travel times as well as changing the validity of assignments generated by some genes.

A simplified example of this method (taken from [14]) is shown below. Suppose we want to recombine the following two parent chromosomes (in which each letter symbolizes a demand to be performed):

(Parent 1): `C D E G B A F`
(Parent 2): `F B G D E C A`

As the crossover begins, we randomly select a number of demands. Suppose the selected demands are B, C and G. To create the first child, we take the order of the three selected demands in parent 1 (which is C, G, B) and order those same demands in parent 2 (which currently has the order B, G, C) in this order while keeping them collectively in the same places in the chromosome:

$$\begin{tikzcd}[row sep=small, column sep=tiny] \text{(Parent 1):} & & C \arrow[d] & G \arrow[d] & & & B \arrow[d] & \\ \text{(Parent 2):} & F & B & G & D & E & C & A \end{tikzcd}$$

We developed a few variations of crossover operators based on this idea.

Mutation

We used a mutation operator that randomly changes a number of demands and resources inside a chromosome. The changes are implemented in such a way that the resulting chromosome is valid – each demand appears in the chromosome exactly once.

Fitness function

A method of grading the solutions is part of the scheduling framework. We use the solution-generation engine for each chromosome in order to generate the schedule, and then use the framework's schedule-evaluation service to assign a quality grade to the chromosome.

3.2 Results

We have applied the GA to many real-world problems, often achieving better schedules, and in shorter times, than those achieved by other methods. Figure 3 shows a typical run of the GA. In this case, there are 620 demands and 88 resources with widely varying skills and locations. The service organization knew that it would be impossible to respond to all the demands within the specified SLA (Service Level Agreement), so some demands had to be scheduled later than their time window permits. Each such assignment is an SLA violation and deducts from the total value of the schedule. This is an example of a situation when a hard constraint needs to be converted into a soft one, namely one that may be violated with an accompanying reduction in solution's value. Here, as in many other places, the GA relies on the flexibility of the underlying framework.

Other cost factors which reduce the total value are overtime hours and total travel time. Since this problem instance allowed all demands to be scheduled (as long as we accept that some are scheduled later than their time window – being late is better than not being there at all!), the schedule's value has a large fixed positive component: the total value of all the scheduled demands.

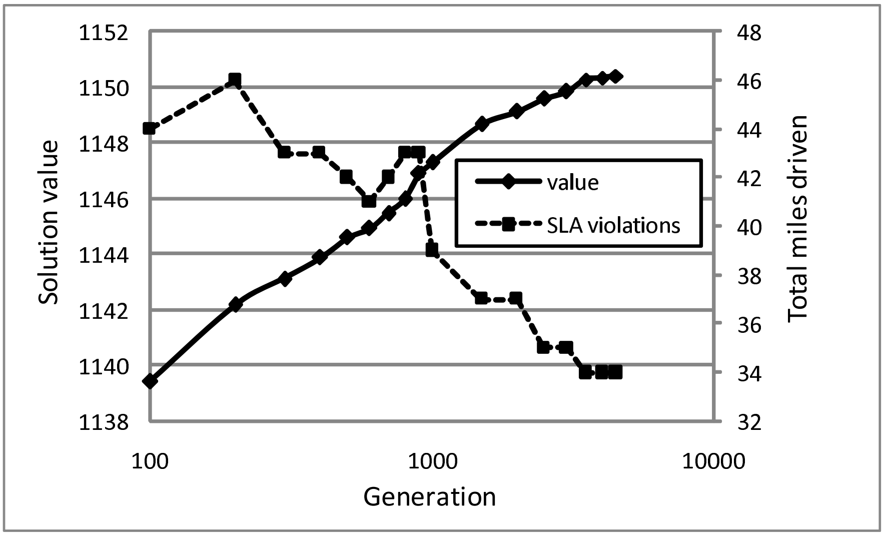

Fig. 3. Typical GA run, showing schedule values and numbers of SLA violations for the best solution in each generation

As we can see in the figure, the semi-logarithmic plot of value against generation number is close to linearity. It is interesting to note that at two points the number of SLA violations increases as the search finds solutions which compensate for this growth by reducing other cost components of the value function, such as mileage and overtime. However, such events are followed both times by a reversal, in which the number of SLA violations drops to its earlier level and even lower. This shows the power of the GA's exploration: having found a better value point, it uses it as an "anchor" from which to look for reductions in costs of SLA violations. After all, it makes sense to expect that as we find solutions which involve less driving there will also be some additional free time. We can then use some of this time to reorder the demands and reach the customers faster. The GA discovers and utilizes such opportunities without being designed with such domain understanding – just using blind, random evolution over suitably designed chromosomes.

4 Ant Colony Optimization

Ant colony optimization (ACO - see [3]) is a family of algorithms inspired by the way ants, which are relatively simple creatures following relatively simple behavioral patterns, exhibit group behavior of great complexity, be it building anthills, constructing live bridges or optimizing the path between a food source and the nest. The mechanism of communication are pheromones deposited by ants according to the circumstances in which they find themselves. These pheromones in turn directly influence the behavior of other ants that happen to find themselves in the same place, which leads to reinforcement of (for the group) positive behaviors and rejection of negative ones. Ant colony optimization algorithms are essentially weighted graph traversal algorithms that take this idea, implement it in software, and optionally enhance it with local search which, as also reported in prior research (see [6] for a VRPTW context), often improves ACO performance. Some local search operators for VRPTW are described in [9], and may be adapted and extended for the field service problem.

4.1 Pheromone Tables

In our pheromone tables, arc (A, B) represents the attraction associated with the scheduling of demand B immediately after demand A. There are also arcs connecting demands to the resource home-base. The pheromone tables are naturally asymmetrical: while $A \to B$ might be highly preferred, $B \to A$ might be very undesirable or even illegal due to the combination of time constraints and geography. We use an individual pheromone table for each resource. We found that using a single pheromone table for all resources is only appropriate when the resources are essentially identical (e.g. VRPTW), and that it leads to slower convergence and lower quality solutions if the resources in the problem instance at hand are heterogeneous (different skills, calendars, home-base locations). Let us give a couple of examples why individual pheromone tables are important:

- Let us have two resources with different skills. For the first one, it makes sense to schedule a remote low-value demand B after demand A because demand B is on the way from A to high-value demand C whose appointment starts too late for it to be scheduled straight after demand A without waiting. Therefore, the arc $A \rightarrow B$ will be marked with a strong pheromone trail and be strongly preferred. However, if the second resource doesn't have the skills to perform demand C but can still work on both A and B, then the arc $A \rightarrow B$ should have a very low value because it represents a lot of travel and lost time for very little reward.

- Let us have two resources with the same skills that can both be assigned to demands A, B and C from the previous example. Let the two resources have different working times, so that one of them is unable to fit all three tasks into the schedule because her work time finishes too soon. Then, again, for one the arc $A \rightarrow B$ should be strongly preferred, and for the other it should be discouraged.

4.2 Algorithm

The algorithm has two major components on two hierarchical levels – the "controller", which defines the high-level workflow of the algorithm and corresponds to the colony with its ability to exhibit complex behavior and learning, and the "builder", which corresponds to ants and builds specific solutions given the environment created by the colony. First, we present the high-level workflow, the "controller":

1. Create and initialize the global pheromone tables. This includes preliminary constraint propagation to determine which demands are candidates for which resources, and which demand combinations are possible in a legal schedule for each resource.
2. For each colony (the colonies can run in parallel) do the following:
   a) Create a local copy of the global pheromone tables.
   b) For each ant (ants within a colony run consecutively) do the following:
      i. Create a solution (see below).
      ii. Update the local pheromone table based on the following criteria:
         - the quality of the solution in terms of value; and
         - the number of scheduled demands.
3. Scan generated solutions for quality based on the following two criteria:
   - $v > (1 - c_1)v^*$, where v is the value of the generated solution, v^* is the best value found so far, and c_1 is a small positive constant; or
   - $n > (1 - c_2)n^*$, where n is the number of demands in the schedule, n^* is the number of demands in the best schedule so far, and c_2 is a small positive constant.
4. Use thus identified "high-quality solutions" to update the global pheromone tables using the same procedure as in step 2(b)ii.
5. Go to step 2 unless stop criteria are met.

Several parts of this algorithm represent refinements of the generic formulation of the ACO to the problem at hand:

- We are using more than one ant colony, and differentiate between global and local pheromone tables. By doing this, we achieve two goals that we judged to be desirable for the algorithm:

  - Our algorithm is naturally parallel in a sense that the colonies are completely independent of each other and can run in different processes, on different processors or even on different computers.
  - By allowing only the solutions of high quality to update the global pheromone table, we intensify exploration in the vicinity of good quality solutions and accelerate convergence. Of course, there is a price we pay for this – we quickly narrow down the search and potentially miss other promising areas. Still, when dealing with large problems (or many smaller ones) in a limited time, this is often a price one needs to be prepared to pay.
- We update the local pheromone tables after each ant. This again speeds up the convergence at the expense of the breadth of search.
- We don't only use the objective function to update the pheromone table, we use an additional heuristic measure which is the number of demands in the schedule. It turns out that this is quite a strong heuristic because the correlation between the number of demands in the schedule and the quality of the solution is very strong. This makes sense, since more tasks indicates better packing within available time and less travel.

Each solution represents routes for all resources. Let us present the algorithm for the "builder" part of the algorithm (a single ant traversal):

1. Choose a random resource. If all resources have already been chosen, go to step 3.
2. For the chosen resource, repeat:
   a) With probability p_b (0.15 in our case), stop with the current resource and go to step 1.
   b) Probabilistically choose the next demand (or resource's home-base) to append to the end of the schedule based on the following criteria:
      - pheromone level associated with the transition;
      - proximity to the demand (taking into account the latest time at which the work on the demand can start); and
      - level of deprivation for the demand (see below).

      The candidate demands are pre-selected by the framework so they can be scheduled at this point of schedule building in a constraint-satisfying way.
   c) If home-base was chosen or no candidate demands remain then go to step 1.
3. Randomize the order of resources for the next step.
4. For each resource in turn, repeat:
   a) Stop with the current resource if there are no more legal options or if the resource has already chosen a home-base and thus ended its schedule.
   b) Probabilistically choose the next demand (or resource's home-base) to append to the end of the schedule based on the same criteria as in the first stage.
5. If local search is being used then we use simple insertion – we insert deprived demands into the schedule in a greedy (i.e. most value increasing) way.

The reason that the resources are selected randomly for schedule building and that there is a possibility for schedule building to be interrupted is to somewhat balance the opportunities for all resources. If we kept the same order or built the full route for each resource every time, the probability would be quite high that the resources that

Table 1. Problem instances used in ACO tests

Name	Resources	Demands	Comment
small	12	46	2 skills, mostly same-site resources
large	88	620	several skills, geographically dispersed resources, lunch break

were scheduled first would always skim the top choices, and the rest would have to be satisfied with whatever remained. As a consequence, the colony would quickly get stuck with the local optima associated with good quality schedules for the first few resources and the schedules of very inferior quality for the rest, while huge parts of search space hiding potentially much better and more balanced solutions would remain unexplored.

The level of deprivation for each demand depends on how often a demand is left out of solutions created by the colony. If a demand often remains unscheduled, then the arcs leading towards it will be prioritized during subsequent runs. The levels of deprivation for all demands are reset when a new colony is started.

In this section, we present the results of using an ant colony optimization algorithm on two instances of the field service optimization problem. The summary of characteristics of the two instances are presented in Table 1.

The ACO results for both instances are shown in figure 4 and figure 5. In both cases we tried four variants of the algorithm based on two algorithm elements:

- *Individual or shared pheromone tables.* The first variant has a separate pheromone table for each resource, while the other has a single pheromone table for all resources. We discuss the reasons why individual pheromone tables are likely to lead to better results in section 4.1. On the other hand, shared pheromone tables lead to quicker iterations and lower memory consumption.
- *Local search or no local search.* In our case, a simple insertion algorithm was used in the local search step – the deprived tasks were taken one by one, and inserted in the most improving place in the schedule (or, more commonly, left unscheduled). The variant with local search could achieve better solution quality with the same pheromone table, while the one without had shorter iterations and could therefore perform more of them.

4.3 Results

Small instance

This small instance is one of the most commonly used within our suite of test data for the field service problem. The best result obtained so far for this instance was generated by the ACO algorithm described here.

Let us take a closer look at figure 4. The figure shows results of a single run of all four algorithm variants on the small instance – each point represents the moment in time that a new best solution was found (and its value).

We see two very different kinds of improvements. One is incremental and barely noticeable visually. These are improvements stemming from rerouting, reordering or

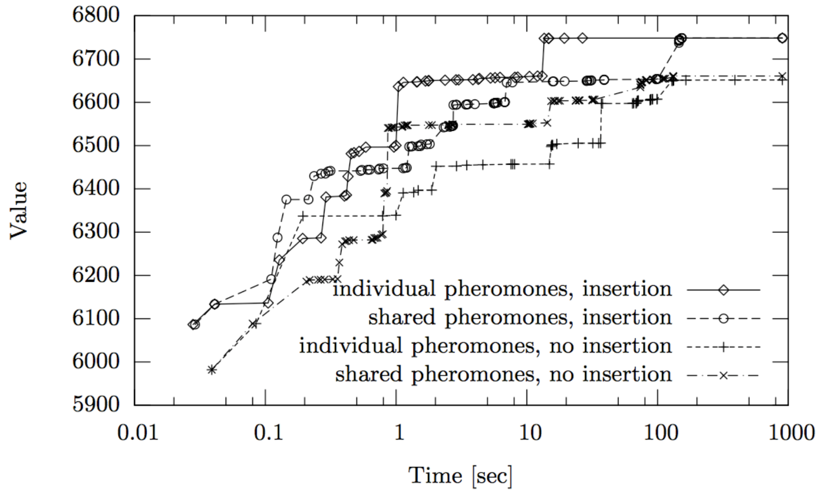

Fig. 4. ACO results for the small instance

replacing one demand with another. The other one is an improvement in which the total number of scheduled demands has been increased. Since typical demand revenue is one to two orders of magnitude greater than typical travel cost, this is expected.

As expected, the two algorithms with local search start with a greater number of assignments and therefore have an immediate advantage. This advantage is maintained throughout most of the run, with a few isolated crossings where one of the algorithms without local search briefly takes over. Local search turns out to be much more important than the individual pheromones for this instance.

In fact, with or without local search the final solution's quality produced with individual pheromones is practically identical to the one with shared pheromones (and happens to be even slightly better in the case of no local search). We can explain this by a very small variety among the resources in this instance – two calendars, three skill sets, only three home-bases, so that five resources are identical in all their attributes. The results seem to be inconclusive with regards to whether using individual pheromone tables is actually useful in this case, since they are associated with (mostly) quicker convergence in algorithm variants with local search, and with slower convergence in those without.

Large instance

Figure 5 shows the results of all four algorithm variants on the large instance. Since this is a more realistic and more complex instance of the field service scheduling problem, we also observe different behavior and draw different conclusions about different algorithm variants.

Initially, we notice that the two instances with local search stay close to each other and are doing better than the two instances without local search, which also remain close together at this stage. This is the time during which the pheromone tables are still largely undeveloped, and the difference between individual and shared pheromone tables doesn't matter that much.

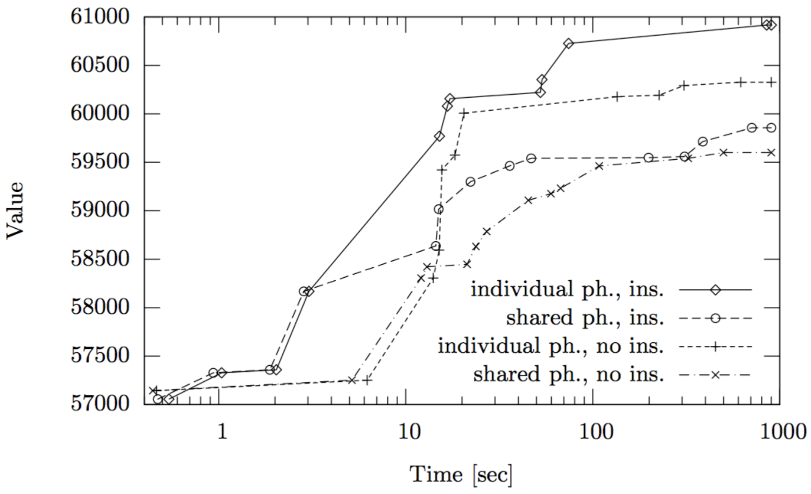

Fig. 5. ACO results for the large instance

However, at between 10 and 11 seconds the situation changes. The ACO with individual pheromone tables but without local search overtakes the one with a shared pheromone table and with local search, and never falls below it again. From that point onwards, the four algorithm variants stay at the same quality ranking. With this more typical instance, the diversity of resources has a dominant role in determining which algorithm variant will do better. Individual pheromone tables perform better than shared, and, given the same pheromone structure, local search is better than no local search.

5 Hybridizing GRASP with ACO

Encouraged by the success achieved by the ACO, we also tried to develop a hybrid algorithm that would enable a GRASP algorithm to take advantage of the learning capabilities of ACO. This is still work in progress, so the presented results are preliminary and should be taken as only indicative of things to come.

The Greedy-Randomized Adaptive Search Procedure (GRASP - see [4]) implementation that we used in this experiment is an optimization method that works in two stages:

1. A solution is constructed using a constructive algorithm in which the decisions are randomized at each stage, with greedier moves (the ones with more positive immediate effect on the objective function) having a greater weight. The specific method in which the moves are weighted is a non-trivial component of the implementation. This stage is called a randomized greedy stage.
2. Local variation operators are used to explore the solutions's local neighborhood in the search space, resulting in ever-increasing changes to the original solution. This stage is called an adaptive stage.

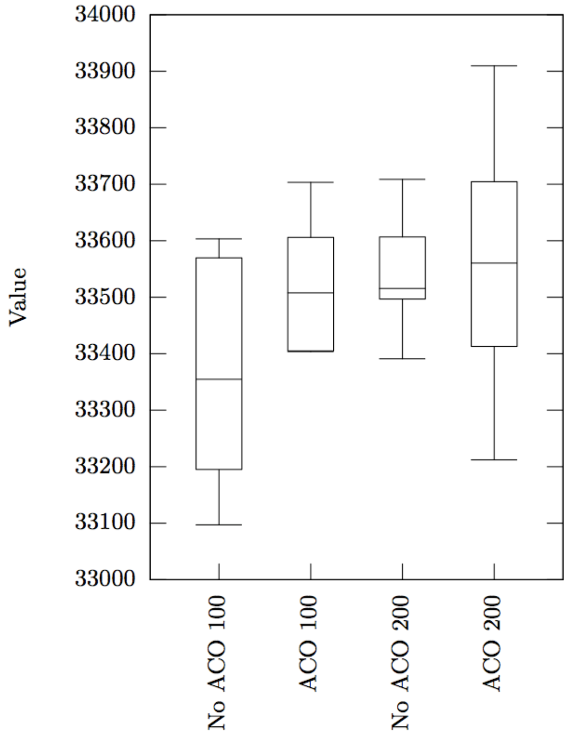

Fig. 6. Box-and-whiskers diagram for results of GRASP and GRASP with ACO after 100 and 200 second runs. Each of the four parts for each algorithm variant and run time represents a quartile over all the runs on the same instance.

Each stage has its own control loop and termination criteria, which need to be adjusted to the problem at hand and fine-tuned during algorithm development.

The main idea of the hybrid algorithm that we are proposing here is to use the ACO pheromones as an additional component of “greediness” during both stages of GRASP. The pheromones are updated after the end of each outer iteration based on the best solution found in each try. Thus, the pheromones collected in previous runs affect the probability distribution over moves in succeeding runs. The intent is that the pheromones will store some global information about the quality of moves, and therefore add a global component to the exclusively local considerations of traditional GRASP. This may be seen as a way of adding learning to a randomized search framework, while controlling its effect by including some forgetting (pheromone evaporation) as well. This algorithm may be compared to the combination of genetic and cultural evolution in human (and some non-human) societies [8].

Our preliminary results are illustrated in figure 6 and certainly show some promise. The diagram depicts all four quartiles over a number of runs on the same problem instance using both two algorithm variants (pure GRASP and GRASP with ACO) over

100[3] and 200[4] seconds. In the shorter run, ACO shows a definite improvement in all four quartiles over pure GRASP. The results over the longer run are a bit more ambiguous: while we notice a quite dramatic improvement in best and a bit less dramatic improvement in median result, we also notice a general increase in spread and therefore a decrease in consistency. Still, given that these are still preliminary results and that a lot of work remains to be done to make the hybrid algorithm more robust, we are greatly encouraged by what we saw and think the approach definitely shows some promise.

6 Human-oriented Schedule Generation

People who use schedule optimization systems include the dispatchers – who view and manipulate parts of the optimized schedule, and technicians – who perform their tasks according to the final schedule. For the application to succeed, these users must accept and trust the software's decisions. Yet, there are several reasons why a purely mathematical approach, striving to get as close as possible to the global optimum, can be perceived by users as failing to deliver a satisfying solution.

One reason for such perceptions is the fact that not all people in the organization agree on the components of the value function, and on the weights which should be assigned to each component. As pointed out above, real-world schedule optimization involves balancing many types of objectives. Some of these objectives may be ranked highly by the regional manager, while the technicians would prefer other objectives to rank more prominently, and the financial staff request raising the weight of still other objectives. A discussion of methods to establish the components and weights of a value function which everyone can agree upon is out of the scope of this chapter, but in this section we present an idea that, in our opinion, is one way to address the collisions of different interests.

Another reason for perceiving the schedule as sub-optimal is the tension between local and global views. A typical example occurs when a technician is driving from area A to area B, and sees another technician driving in the opposite direction - from area B to area A. Both technicians may think that a better schedule would have been to exchange tasks between them, so that the technician who has just completed a task in area A would remain there to start the next task in that area, and similarly both tasks in area B should be performed by the other technician. According to this view, the schedule which was assigned to them leads to longer and more expensive travel while reducing productivity and "wrench time" (time spent on the actual work). In practice, we almost always find that such a "strange" schedule is actually better: for example, even if both technicians have the required skills for these tasks, there may be yet another task waiting in area B which requires the first technician, so this technician would have had to drive to area B anyway, and it's just a matter of finding the best time to do it. There are many other possible reasons that justify this perceived "waste of time", but they all have something in common – they are invisible to the technicians that do not see the whole picture.

[3] With statistical significance of the observed difference at $p < 0.05$ level.

[4] With statistical significance of the observed variance at $p < 0.01$ level.

We have found that while it is possible to explain the rationale behind such scheduling decisions so that the technicians at least outwardly accept that they were given a good schedule, it is also possible to significantly reduce such complaints by adding a few mechanisms into the optimization process. One way is to introduce low-weight components into the value function or the heuristic that slightly reduce the value of solutions which give rise to such complaints. When this is not possible, we can add a “post-processing” phase to the optimization process that identifies such occurrences and uses local search to find friendlier alternatives in the solution’s neighborhood in the search space, thus hopefully avoiding the problem while still maintaining most of the solution’s value. Even though some service organizations are perfectly willing to accept a modest decrease in the solution value in order to balance such local points of view with global considerations, we have found that these mechanisms typically do not noticeably reduce the value of generated solutions (as expressed by the global value function), and only add a modest increment to the execution time. The same approach can be used to address tensions between solution characteristics given different priorities by different parts of the organization: locally searching for a solution whose value is very close to the best solution found, while avoiding situations where one objective seems to have taken over without any consideration for the other objectives.

7 Conclusions

In this chapter, we describe a wide variety of VRPTW-like optimization problems encountered in field service scheduling, and introduce a conceptual framework for categorizing these problems by the relevant attributes of major entities (e.g. demands, resources, customers, equipment), by the constraints defining acceptable solutions, and by the components of the value function which together define the quality of solutions. We also describe a framework for modeling problems of this family and for separating search algorithms from the services required for efficient constraint propagation and for building and evaluating partial solutions. These conceptual and algorithmic frameworks have enabled us to build, evaluate and hybridize many different search algorithms, including approaches from the fields of mathematical programming, clustering, temporal constraints, local search, and population search such as GA and ACO.

We note that the algorithms presented here are not entirely robust: that is, a few parameters need to be determined according to the problem’s characteristics. They include, for example, the size of the population and the number of crossover operations in each iteration for the GA algorithm, and probability and pheromone-update parameter values for the ACO algorithm. However, since most heuristic approaches require some parameter setting as well, this need for fine-tuning is not a limitation that is unique to the approaches described here.

A workable field service optimization system needs quite a few additional capabilities which affect its design and applicability. Among these, we mention **scalability** – the ability to handle problems beyond the abilities of a single computer while maintaining the visibility of the ”best-known” solution across all participating computers; **real-time optimization** – the ability to find good solutions immediately after new information is received, and to continue optimization concurrently with other updates and with actions by the human users which change the schedule; and **optimization across multiple time horizons** - the ability to plan ahead for the next week based on a statistical prediction of demand (when no tasks are yet known) and for next month based, additionally, on statistical prediction of capacity (since vacations are not yet known), while allowing the interchange of information and decisions between the different planning processes.

The practice of developing and extending commercial optimization software bears significant similarities with, as well as significant differences from, academic research. We gratefully acknowledge our debt to the academic research which contributed many of the methods we use, and our debt to the cooperation of the service organizations who look to these mathematical and algorithmic techniques as a tool to get the job done – to deliver service effectively and efficiently.

7.1 Remarks

The authors would like to stress that the contents of this chapter are not a description of the algorithms used by ClickSoftware Ltd., the commercial software vendor with which all authors are or have been associated. ClickSoftware's algorithms are proprietary and involve a combination of methods and building blocks. Our intent here is to report some of the research we've performed in order to improve our understanding of the problem and of potential algorithms, leveraging our access to real-world problem instances and to comments made by business organizations that evaluated the solutions generated during these experiments. We gratefully acknowledge the help, support and contributions which our research received from ClickSoftware management and from our colleagues.

References

1. Beniaminy, I., Nutov, Z., Ovadia, M.: Approximating interval scheduling problems with bounded profits. In: Proceeding ESA 2007, Eilat, Israel, pp. 487–497 (2007)
2. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
3. Dorigo, M., Stutzle, T.: Ant Colony Optimization (Bradford Books). MIT Press, Cambridge (2004)
4. Feo, T., Resende, M.: Greedy randomized adaptive search procedures (1995)
5. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: SODA 2006: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 611–620 (2006)
6. Gambardella, L.C., Taillard, E., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical report, IDSIA, Lugano, Switzerland (1999)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, Reading (1989)
8. Jablonka, E., Lamb, M.J. (eds.): Evolution in Four Dimensions: Genetic, Epigenetic, Behavioral, and Symbolic Variation in the History of Life (Life and Mind: Philosophical Issues in Biology and Psychology). MIT Press, Cambridge (2006)
9. Larsen, J.: Parallelization of the Vehicle Routing Problem with Time Windows. PhD thesis, Department of Mathematical Modeling, Technical University of Denmark (1999)

10. Louis, S.J., Yin, X., Yuan, Z.Y.: Multiple vehicle routing with time windows using genetic algorithms. In: Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A. (eds.) Proceedings of the Congress on Evolutionary Computation, Mayflower Hotel, Washington D.C, vol. 3, pp. 1804–1808. IEEE Press, Los Alamitos (1999)
11. Mitchell, G.G., O'Donoghue, D., Barnes, D., McCarville, M.: GeneRepair - a repair operator for genetic algorithms. In: Rylander, B. (ed.) Genetic and Evolutionary Computation Conference Late Breaking Papers, Chicago, USA, July 12–16, pp. 235–239 (2003)
12. Nutov, Z., Beniaminy, I., Yuster, R.: A (1-1/e)-approximation algorithm for the generalized assignment problem. Oper. Res. Lett. 34(3), 283–288 (2006)
13. Spieksma, F.: On the approximabilty of an interval scheduling problem. Journal of Scheduling 2, 215–227 (1999)
14. Stein, R., Dhar, V.: Satisfying customers: Intelligently scheduling high volume service requests. AI Expert 12, 20–27 (1994)
15. Tan, K.C., Lee, L.H., Zhu, K.Q., Ou, K.: Heuristic methods for vehicle routing problem with time windows. Artificial Intelligence in Engineering 15(3), 281–295 (2001)
16. Weigel, D., Cao, B.: Applying GIS and OR techniques to solve Sears technician-dispatching and home delivery problems. Interfaces 29(1), 113–130 (1999)
17. Whitley, D.L., Gordon, V.S., Mathias, K.E.: Lamarckian evolution, the baldwin effect and function optimization. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) Parallel Problem Solving from Nature – PPSN III, pp. 6–15. Springer, Berlin (1994)