# An Evolutionary Approach to Practical Constraints in Scheduling: A Case-Study of the Wine Bottling Problem

Arvind Mohais, Sven Schellenberg, Maksud Ibrahimov,
Neal Wagner, and Zbigniew Michalewicz

**Abstract.** Practical constraints associated with real-world problems are a key differentiator with respect to more artificially formulated problems. They create challenging variations on what might otherwise be considered as straightforward optimization problems from an evolutionary computation perspective. Through solving various commercial and industrial problems using evolutionary algorithms, we have gathered experience in dealing with practical dynamic constraints. Here, we present proven methods for dealing with these issues for scheduling problems. For use in real-world situations, an evolutionary algorithm must be designed to drive a software application that needs to be robust enough to deal with practical constraints in order to meet the demands and expectations of everyday use by domain specialists who are not necessarily optimization experts. In such situations, addressing these issues becomes critical to success. We show how these challenges can be dealt

Arvind Mohais · Neal Wagner
SolveIT Software Pty. Ltd., Level 1, 99 Frome Street,
South Australia 5000, Australia
e-mail: {am,nw}@solveitsoftware.com

Sven Schellenberg
SolveIT Software Pty. Ltd., Suite 201. 198 Harbour Esplanade,
Docklands, Victoria 3008, Australia
e-mail: ss@solveitsoftware.com

Maksud Ibrahimov · Zbigniew Michalewicz
School of Computer Science, University of Adelaide,
South Australia 5005, Australia
e-mail: {maksud.ibrahimov,zbigniew.michalewicz}@adelaide.edu.au

Zbigniew Michalewicz
Also at Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland and Polish-Japanese Institute of Information Technology, ul. Koszykowa 86, 02-008 Warsaw, Poland

with by making adjustments to genotypic representation, phenotypic decoding, or the evaluation function itself. The ideas presented in this chapter are exemplified by the means of a case study of a real-world commercial problem, namely that of bottling wine in a mass-production environment. The methods described have the benefit of having been proven by a full-fledged implementation into a software application that undergoes continual and vigorous use in a live environment in which time-varying constraints, arising in multiple different combinations, are a routine occurrence.

## 1 Introduction

The application of evolutionary algorithms (EAs) to real-world problems brings us face-to-face with some challenging issues related to the time-varying nature of such problems. These issues are over and above the fundamental problems being solved, and in fact, they can be so critical from the standpoint of creating a robust and truly usable software application that their consideration significantly alters the approach taken to solve the problem.

From a pure-problem perspective, EAs have been used for quite a long time to solve scheduling problems [1, 2, 3]. A general scheduling problem requires that we make decisions on how to assign a number of jobs to a fixed set of machines able to perform those tasks. The assignment must be done in such a way as to minimize the amount of time required to complete all of the tasks (referred to as the makespan). There are many variations to this problem. For example, the processing of a job may require that it passes through multiple pre-defined phases, each possibly taking place on a different machine. All of the variants however are equivalent to the NP-Complete problem of *job shop scheduling* [4]. This of course makes it virtually unsolvable in a reasonable amount of time using currently known techniques. Thus, a usual approach is to look for an approximate solution using EAs.

The essence of a straightforward evolutionary approach is to represent a candidate solution as an ordered list of (machine, job) pairs, with the natural order of the list being representative of the sequence of job execution. Using a well-constructed fitness function that embodies the problem constraints and parameters, and perturbing the representation using simple operators will usually guide the population to a good solution.

This kind of approach is quickly seen to be insufficient when it comes to creating a fully-fledged solution that is to be used in a real-world business environment to manage the day-to-day scheduling needs of a large enterprise. There are many issues that intrude on the purity of a simple approach such as the one described above. These issues are usually critical and require careful consideration so that the business needs can be satisfied in a manner compatible with the architecture of the EA.

The issues in question arise because of the dynamic nature of real-world problems. One hardly ever encounters a situation that is static for any significant

period of time. In everyday business life, things change almost constantly. The number of orders to be scheduled in a factory changes, the run rate of a machine varies due to environmental factors, the quantity of a particular item ordered is modified after it has already been allocated, machines break down, delivery trucks arrive late, and so on.

In this chapter we examine examples of such practical time-varying issues that arise in a specific scheduling problem, namely, what we will refer to as the Wine Bottling Problem. We show how an EA along with supporting software components can, by proper design, effectively deal with these issues.

The rest of this chapter is organized as follows. We start with a classification of time-varying issues into three categories, alongside a brief literature review of work in each category in Section 2, and of scheduling problems in Section 3. Following this, in Section 4, the main case study of this chapter, an industrial wine bottling problem, is elaborated in sufficient detail to enable the reader to more easily visualize the kinds of problems being addressed. We go on to describe the real-world business issues that had to be considered and resolved in order to build a solution around an EA core in Section 5. The design and construction of the EA itself is fully specified in Section 6 and alongside this we illustrate how the various time-varying issues were resolved in relation to EA components.

## 2 Dynamic Optimization Problems

The dynamic nature of real-world optimization problems is well known, but a closer examination reveals a few different aspects of the problem that can described as *dynamic*. In this section we introduce a classification of dynamic optimization problems into three categories, characterized respectively by:

1. Time-varying objective functions.
2. Time-varying input variables.
3. Time-varying constraints.

There is a large body of EA research literature that addresses this dynamic property of such optimization problems, and we will undertake a brief review of work in each of these areas. It will be noticed that, while there is an abundance of work on problems fitting into the categories of time-varying objective functions and time-varying input variables, there are relatively few published reports dealing with dynamic optimization problems of the third kind, that is one that deal with time-varying constraints.

### 2.1 Time-Varying Objective Functions

These are problems in which the shape of the fitness landscape varies with time, that is, the location of the optimum value is continually shifting. The literature shows that there has been a considerable amount of research done

on this kind of problem from the early days of evolutionary methods, to the present [5, 6, 7, 8, 9]. Some are based on artificially constructed objective functions and others on real-world applications. In many published reports, situations were set up wherein during a single run of an EA, the location of the optimum would move, and researchers were particularly interested in developing EAs that could detect that there has been a change in the optimum and continue to alter their search to find the new optimum.

Some examples of the earliest papers related to dealing with time-varying objective functions using EAs are [10] and [11]. These papers report on attempts to track optima in fluctuating, non-stationary environments (objective functions). Pettit and Swigger studied environments that fluctuated stochastically at different rates and they envisioned applications to areas such as voice recognition and synthesis.

Grefenstette extended the work of Krisnakumar [12], by pursuing ideas related to maintaining diversity. That work considered an abstract problem that used an optimization surface that would change randomly every 20 generations of the genetic algorithm, with new peaks appearing and previous optima being replaced by new ones. Although not addressed as an application in that paper, reference was made to real-world dynamic problems such as optimizing the fuel mixture for a jet engine based on the measurement of engine output. This problem is dynamic because of both slow and rapid changes. The components of the engine slowly degrade over time and hence, slowly alter the function being optimized, and it is also possible to experience a rapid change due to the sudden failure of a component. It was clear that the intent of the work was to serve as a foundation for dealing with these types of problems.

Different authors have worked to create standard test suites that can be used to study time-varying objective functions. For instance, Branke [13] has published a test function called the moving peaks benchmark (MPB) problem. This is an abstract function that represents a landscape with several peaks. Each peak is defined by a height, a width and a position in the xy-plane. The function is dynamic because these three characteristics of each peak are altered slightly as time progresses. It is easy to imagine this function surface slowly changing with time, with the xy-coordinates of the peak representing the global optimum slowly moving, and also possibly, the global optimum shifting from one peak to another.

The case study that will be considered in this chapter does not fall into the category of dynamic problems discussed in this section. When considered from a broad perspective, the wine bottling problem can be thought of as having a time-varying objective function, but it is not one that changes during the course of an EA run, rather the objective function changes with each run of the application as business conditions change.

## 2.2 Time-Varying Input Variables

These are problems in which the input data being processed in the optimization scenario changes from day to day, or in principle from run to run of the EA. For example, if we are interested in optimizing the production schedule of a factory, then each day, or possibly each minute, the set of customer orders to be allocated and sequenced on machines will change. This is because in a live business environment, as time passes, old orders are completed, and new ones arrive.

In [14], Johnston and Adorf describe how artificial neural networks can be used to solve complex scheduling problems with many constraints and report the real-world problem of scheduling the assigned usage of the NASA/ESA Hubble Space Telescope was solved using an application based on these techniques.

Chryssolouris and Subramaniam [15] explore the dynamic job shop problem where multiple criteria are considered and jobs may be processed by alternate machines. Although they recognized the lack of work on dynamic constraints such as random job arrival, in terms of input to the EA, their work primarily focused on multiple job routes.

Madureira, Ramos and Carmo Silva [16] investigated using a genetic algorithm to study how to produce schedules in a highly dynamic environment where new input variables require continual re-scheduling. Their work was based on an extended job-shop in a dynamic environment with simple products and such that require several stages of assembly. Another example of recent work done in this area include [17].

## 2.3 Time-Varying Constraints

These are problems in which the constraints of the environment within which a solution must be found change from day to day. These varying constraints add an additional level of complexity to the problem because a good EA approach must be able to operate equally well regardless of how many of these constraints are in place, and in what particular combination they occur. For an example of a time-varying constraint, consider a manufacturing environment in which the problem is to generate a production schedule for a set of jobs. After a schedule has been produced and passed to the factory floor for execution, it is customary to *freeze* a number of days of the schedule. This means that it is mandated that for the specified number of frozen days, the schedule will remain fixed. Any subsequent revised schedules should not alter the contents of the current schedule for these days. In this context, whenever a new schedule needs to be produced, it is a requirement that it mesh seamlessly with the frozen period of the existing schedule. In general, we do not know what the current schedule might be at any given moment in time,

yet we must create an EA that generates a new solution that matches up with the existing one, and still produces an optimal result for the long term. As new schedules are produced every day, the contents of the frozen period varies, yielding a time varying constraint.

Good examples of the types of problems that can be categorized as time-varying constraints can be found in [18], wherein Jain and Elmaraghy describe circumstances that require the regular generation of new production schedules due to uncertainties (both expected and unexpected) in the production environment. They touch on typical examples such as machine breakdowns, increased order priority, rush orders arrival, and cancellations. All of these issues are also considered in this chapter, from the perspective of an integrated EA-based software solution. There are many other examples of works regarding this type of problem, for instance in [19, 20, 21].

## 2.4 Constraints Encountered in Practice

Based on our experience in solving real-world optimization problems for commercial organizations, we have found that the type of problems commonly experienced in industry belong to the second and third categories. Interestingly we have found that the vast majority of published research in EAs addresses the first category and to a lesser extent the second category. However, dynamic optimization of the third kind, i.e., where the problem involves time-varying constraints, although well-recognized in other domains, has been the subject of relatively few investigations in the EA literature. This observation is especially true when we extend our search to fully-fledged application of an EA to a dynamic real-world problem. Figure 1 illustrates some examples of issues that typically arise in real-world problems.

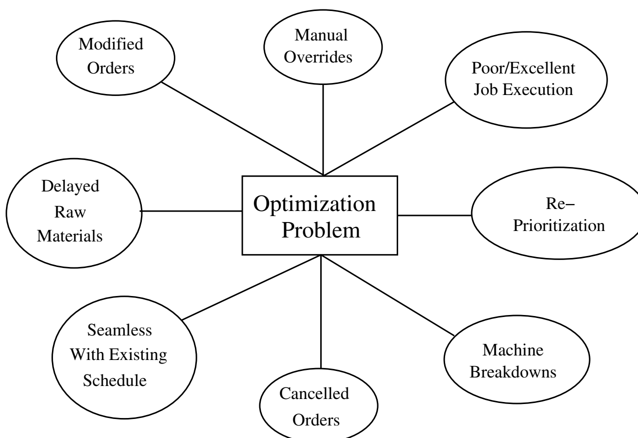


**Fig. 1.** Typical Issues in Real-World Problems

## 3 Scheduling Problems

The case study examined in this chapter, the wine bottling problem, is presented as a scheduling problem. It is a textbook example of a real-world industrial problem whose solution requires finding a production schedule to run a factory, or multiple factories. Essentially, given a number of tasks that need to be performed, we need to determine the best way of executing the required work, in terms of deciding where to assign work, and at exactly what time, and in what sequence the individual tasks should be carried out. Before getting into the wine bottling problem properly, in this section, we first take a brief look at some papers that give the reader a cross-section of examples of the application of EAs to scheduling applications, both classical academic problems, as well as real-world problems.

The Job-Shop Scheduling problem is a canonical example of a scheduling problem. It is one of the most difficult combinatorial optimization problems, and as is the case with most scheduling problems, it is NP-complete [4]. A very good survey of job-shop scheduling problems and different solution representations is given in [22]. [1] discusses the application of genetic algorithms to the job-shop scheduling problem. Wang and Zheng [3] solve it using a modified genetic algorithm. In [2], a simulated annealing approach to tackle the job-shop scheduling problem is discussed.

Yamada and Reeves [23] proposed an EA to solve a scheduling problem called the permutation flow shop problem. They combined stochastic sampling and best descent methods into one unified method to reach effective results. Their work was based on a method described by Nowicki and Smutnicki in their tabu search algorithm for the flow shop problem [24].

There are also many published reports of the application of EAs to solve scheduling problems in various domains of application. Marchiori and Steenbeek in [25] developed an EA for the real-world airline crew scheduling problem. Results of the real-world benchmark instances were compared with the results produced by the commercial systems and produced effective competitive results.

Ponnambalam and Reddy [26] developed a multiobjective hybrid search algorithm for lot sizing and sequencing in flow-line scheduling. The main idea is in the memetic type of algorithm that combines genetic algorithm with local search procedure.

A practical problem of optimal oil production planning was discussed by Ray and Sarker [27]. Production is based on several oil wells, from which a crude oil is extracted by means of injection of a high pressure gas. The goal of the problem was to optimize amount of gas needed to be used in each of wells to maximize output of oil taking in account a limited amount of gas each day. Single objective and multiple objective versions of the problems were considered.

Burke and Smith [28] investigated a real-world problem that addressed the maintenance problem of a British regional electricity grid. They compared the performance of a proposed memetic algorithm with other methods.

Martinelli shows the optimization of time-varying objective functions using a stochastic comparison algorithm in [29]. Values of the time-varying functions are known through estimates. Noise filtering was introduced to decrease probability of wrong moves.

Tinós and Yang introduce a self-organizing random immigrants scheme for algorithms in dynamic environments in [30]. Newly immigrated individuals are held in the subpopulation for some time until they develop good fitness values.

Yang discusses in his chapter [31] a memory scheme approach as a method of improving the performance of EAs for dynamic environments. Two kinds of memory schemes are described: direct and associative. These schemes are applied to genetic algorithms and on univariate marginal distribution algorithms in dynamic environments.

Schönemann [32] considers the application of evolutionary strategies for numerical optimization problems in dynamic environments. The main parameters of evolutionary strategies for problems in dynamic environments are presented and performance measures are discussed with advantages and disadvantages of each of them.

## 4 Scheduling Case-Study: Wine Bottling

Wine manufacturing and EAs go particularly well together. From the very starting point of planting grape vines and reaping the mature fruit, all the way through the crushing of the grapes, management of bulk tank movements during the fermentation process, to bottling of the finished product and sales, the wine manufacturing industry is a rich source of real-world application areas. Many of these problems are based on classical optimization problems and on that basis alone are quite difficult to solve. EAs, of course, by their very nature are natural takers for these kinds of challenges. In this section, we examine what is involved in one of the wine manufacturing steps just described, namely the bottling of the finished product.

Before getting to the point where wine is bottled into a finished product, the liquid would have gone through a series of fermentation and other processing steps. We will assume that we are at the point where the liquid is in a finished, consumable state, and is residing in a bulk storage tank, which could be anywhere from several tens of thousands of liters, up to more than one million liters in volume. This bulk liquid remains in storage awaiting the bottling process wherein it is pumped into a bottling factory and put into consumer size bottles, with a typical volume of less than one liter each.

A bottling factory houses several bottling *lines*. These are machines that are connected to intermediate feeding tanks that contain finished wine, and

are used to transfer the wine into glass bottles and hence produce the items that everyday consumers are accustomed to purchasing. The bottling lines also take care of related tasks such as capping the bottle with a screw cap or a cork, applying labels to the bottle, and packaging the bottles into cartons. Each line is capable of bottling a subset of the types of finished wine products manufactured by the wine company.

These two elements, the bulk wine liquid, and the bottling lines, constitute the basic working elements of the wine bottling problem. The bottling process is illustrated in Figure 2. Customer orders determine which bulk wines are put into which bottles and the times at which that is done. The wine company receives orders for particular finished goods from their clients and it is those orders that must be carefully considered in order to determine the best way of running the bottling plant. In an ideal situation, customers place their orders with sufficient notice to ensure timely bottling of their goods.

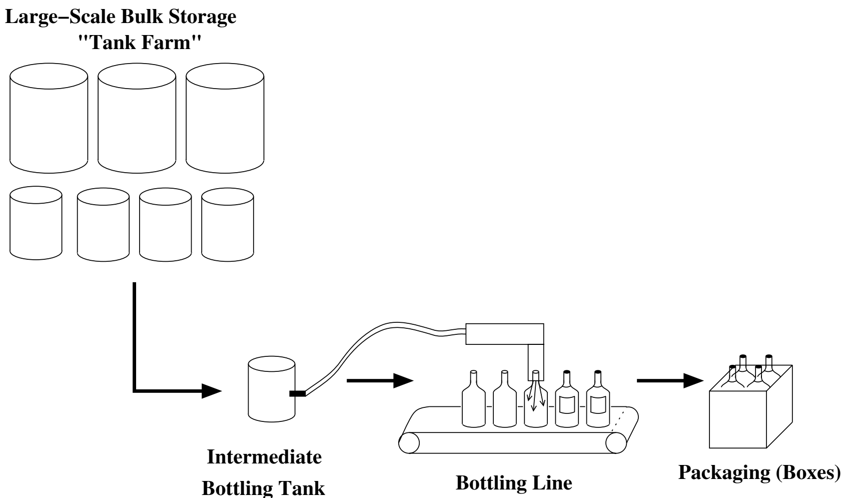


**Fig. 2.** The Wine Bottling Process

The problem is to determine a sequence of orders to be carried out on each bottling machine (hence classifying it as a scheduling problem) such that optimal use is made of the company's resources, from the point of view of making maximum profit, and also maximizing customer satisfaction. Hence a good schedule will minimize production costs and at the same time ensure that orders are produced in full, sufficiently before their due dates. This is the fundamental part of the wine bottling problem: deciding how to schedule production so as to make the best use of limited resources.

Wine making, as is the case in a multitude of other real-world applications, is a highly variable and complex business, and what, from the point of view of using an EA, would otherwise be considered a straightforward scheduling

application, gives rise to a series of significant challenges. These practical day-to-day business considerations are what make this problem highly dynamic, and we are led to classify the problem as a dynamic optimization problem of the third kind, due to the fact that and many of the issues are based on time-varying constraints.

## *4.1 Basic Constraints in Wine Bottling*

Before getting into the business intricacies that give rise to the time-varying constraints, we will first consider some more basic issues that affect the scheduling problem:

**Due dates:** When a customer places an order, the sales department will assign to it a due date that is acceptable to the customer and which should also be realistic taking into account the size of the order and available resources at the bottling plant. Sometimes, for various reasons, due dates are unrealistic, but nevertheless, the scheduling application needs to employ techniques that strive to maximize the number of orders that are *delivered in full on time* (this expression is sometimes abbreviated as DIFOT). In cases where it is impossible to have orders delivered on time, the algorithm must strive to minimize the delays incurred on each order. This is a very important objective in the application, since it is of utmost importance that customers should not be displeased due to late delivery of orders.

**Bulk wine availability:** Some orders may need to be inevitably delayed due to the fact that the bulk wine needed to fill the bottles may not yet be ready. This could easily happen since the process of creating wine is quite variable, and batches may not have responded to the fermentation process as expected, and could require additional processes to get the wine to the required specification and taste. There are a number of other processes involved in the preparation of the bulk wine that can lead to delays, such as filtration and temperature stabilization.

**Dry Materials:** In addition to the liquid wine, there are a few dry goods that are required to produce a finished bottle of wine. First there is the glass bottle, then there is its covering which may be a screw cap or a cork, and several other items such as labels, of which there may be several (for the front, back and possibly neck of the bottle), and foils and wire hoods for sparkling wines. If an order is scheduled at a particular time, then to proceed, the appropriate amounts of each dry good must be ready for installation into the bottling machine. Hence it is a requirement that the optimization software checks the availability of these materials in order to produce feasible schedules.

**Job run lengths:** It is inefficient to have machines frequently changing from one type of bottling job to another because this incurs set-up and take-down time and reduces the overall utilization of the machine. Hence, the scheduling

algorithm must attempt to group similar orders for sequential execution so this type of inefficiency is avoided.

**Wine changeovers:** Wines are categorized broadly in terms of their color, there is red, white, and rose (pink). Below this level of classification, there are more detailed distinctions such as sweet red, dry red, aromatic white, full-bodied white, sparkling red, sparkling white, fortified wine, and others. Even for a very large wine company, it may not be the case that there is sufficient production volume to justify dedicating individual machines exclusively to the production of one type of wine. Hence the reality is that the same machine must be used at different times to bottle different types of wine. When a bottling line finishes working with one type of wine and switches over to another type, this is referred to as a *wine changeover*. Bottling lines must be cleaned during changeovers, to varying degrees, depending on the nature of the change. Certain types of wine changeovers are undesirable and need to be avoided where possible by the optimization algorithm. For example, if we are changing over from a run of white wine to a run of red wine, then a relatively brief cleaning is required, since residual amounts of white wine entering into red is not much of a problem. However, the reverse situation where we go from a run of red to a run of white requires a very extensive cleaning process, including sterilization. Minute amounts of red wine entering into a run of white wine are likely to cause a lot of damage.

**Other changeovers:** Although wine changeovers incur the most time, there are a variety of other changeovers that could happen, even within a run of the same color wine. Each different finished good that is being produced requires a particular size and type of glass bottle, as well as a particular type of covering, which may be a screw cap or a cork, of which there are several varieties, and unique labels for each brand of wine. The physical re-configuration of the bottling line hardware to accommodate these items requires a strip-down time to remove the items used by the previous run, and a set-up time to insert the new items required for the next run. The optimization algorithm needs to try to minimize these changeovers as far as possible by appropriately grouping orders.

**Bottling line availability:** Some industries use machines that are kept in operation continuously. This is sometimes the case in large-scale wine companies, but for only limited periods of time. In most scenarios, bottling lines have typical hours of operations that correspond to an average workday, which may be for example 8:00am to 6:00pm. The optimization algorithm needs to consider the availability of each machine in order to make appropriate scheduling decisions since one of the critical factors affecting the evaluation of a proposed schedule is the number of orders that are satisfied on time. In computing the amount of time that a job will take to perform and making a decision about where to assign it, machine availability must be factored in.

**Routings:** Each product can be bottled on a number of different lines. The choices are usually a proper subset of all of the machines. Although the same product could be bottled on a few different machines, the performance characteristics on each machine is likely to be different, sometimes significantly so. Set-up and strip-down time could vary, and also the speed at which the bottles are processed (called the *run rate*) could vary significantly. Each possible assignment of a finished good to a bottling line, together with its associated performance data, is referred to as a *routing*. The optimization algorithm must strive to choose the best routing to use under the circumstances. If all else is equal, then naturally the fastest routing would be chosen. However, this is often not the case, as some lines may be heavily loaded thus preventing such a choice from being made. In order to achieve better load balancing between the machines, alternate routings may have to be used.

## 4.2 The Software Solution

Our work on the wine bottling problem resulted in a full-featured piece of software, built around a core EA that deals with all of the above listed issues and creates feasible, optimal schedules for satisfying customer orders for wine. The application was launched for a major global wine company and it experiences heavy daily usage at international sites. It is a cornerstone of their scheduling department, and we think it is a good example of an integrated EA serving robustly in prime time. The main screen of the application is shown in Figure 3 with all confidential client information removed. The area of horizontally adjacent rectangles shown in the upper portion of the picture is a graphical representation of the schedule produced by the software. Each rectangle corresponds to a customer order that has been assigned to a bottling line at a particular time. The length of the rectangle is indicative of the amount of time required to execute the job and its color coding lets the user know at a glance the type of wine being produced. The graphical interface also allow the user to drag and drop the rectangles to make manual adjustments to the schedule produced by the optimization algorithm.

Figure 4 shows the configuration interface of the software application that allows the user to enter parameters that define the availability of each bottling line. This interface needed to be completely flexible in terms of allowing the user to specify any possible set of available and unavailable times for each line. In this application, the user is allowed to specify a typical weekly timetable, for example a machine may be available from Monday to Friday from 8:00am to 6:00pm, and unavailable otherwise. It also allows the user to specify periods of time in which there is a deviation from this typical schedule. For example, ahead of a festive season, it might be desirable to make the machines work longer hours, and possibly on weekends. This interface also allows the user to allocate periods of time for regularly scheduled downtime for maintenance of the machines.

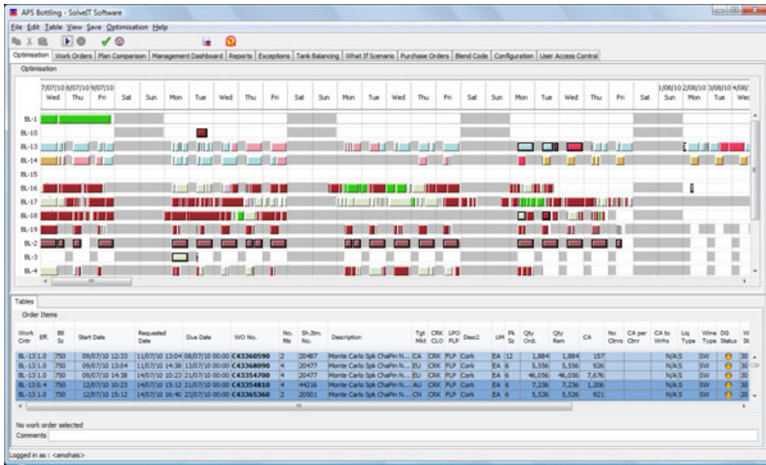

**Fig. 3.** Main Screen of Software Application

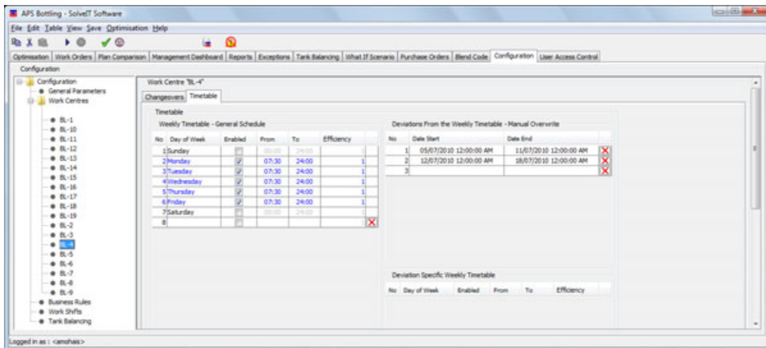

**Fig. 4.** Machine Availability Interface

Figure 5 shows a material requirements dialog that is part of the software application. For any given order, the user may view the materials required to produce the required number of bottles of wine, such as caps, corks, labels, and so on. The application provides the user with information on the availability of each material based on knowledge of opening inventory stock that it uses for a running simulation over the time frame of the production schedule. Additional knowledge of still-open purchase orders for additional materials, and also knowledge of the lead times of various suppliers of these materials allows the software's underlying algorithm to make appropriate scheduling decisions so as to best ensure that orders are sequenced in a feasible manner. If scheduled too early, at a time when the materials would not be available,

**Fig. 5.** Material Requirements Interface

an order would have to be passed over, possibly leading to other disruptions in the schedule and hence in the factory itself.

## 5 Time-Varying Challenges in Wine Bottling

In this section, we will look at some business requirements that lead to time-varying constraints that had to be addressed in the software. Here we will only consider the issues. Their actual solution, including algorithmic details, will be covered in a later section.

**Manual assignments:** There are various scenarios in which the human scheduler would need the ability to override the schedule produced by the EA. For instance, it might be that some of the information used as input by the algorithm such as the availability of dry goods and expected delivery times of raw materials, which are all imported automatically from the company's Enterprise Resource Planning (ERP) system, may for some reason be inaccurate. Hence the schedule that is created may not be feasible and would need to be corrected by the human scheduler. Another typical scenario is that a very important customer makes a late, but urgent request for a large quantity of wine. Even if this causes severe disruptions to the smooth running of the bottling plant, this type of request is usually accommodated due to the high value placed on some customers, and the importance maintaining a good business relationship with them. In this kind of situation the unusual placement of that urgent order in the production sequence would create a different optimization problem – a much more constrained one that would be pose greater difficulty for the EA to solve. The level of difficulty would increase even more dramatically when there are multiple such constraints. The software application that we developed had to be flexible enough to allow its built-in EA to work in such a way that it could actively seek out an optimal solution that satisfies the constraints described before, but at the same time allow inefficient manual overrides dictated by a human operator to co-exist

with the otherwise optimal solution. Another way of looking at this is that the structure of the search algorithm had to be flexible enough to find locally optimal solutions in restricted neighborhoods of the overall search space, with those neighborhoods being defined by user-specified manual assignments.

The application presents the schedule with both graphical and tabular representations, each of which may be intuitively manipulated by the user, simply by dragging and dropping graphical rectangles, or rows of a table. This provides the capability of altering the sequencing of orders to suit a human user's preference. Additionally, as shown in Figure 6, there is a manual assignment interface that is accessed on a per-order basis and allows the user to manually specify all details about when and where a particular job should be scheduled.

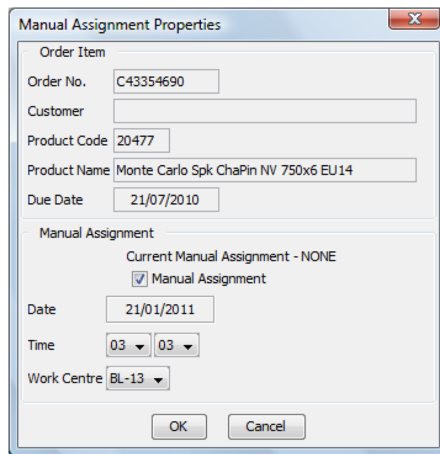


**Fig. 6.** Manual Assignment Interface

**Machine breakdowns:** From time to time, a bottling line will break down and become unavailable for use. It may be that a solution found by the optimizer previously would have been planned around that machine being available during a period of time that has now become unavailable due to the breakdown. The software must be flexible enough to repair the previous solution to take into account the breakdown. A very simple initial solution to this could be to merely shift the sequence of orders previously assigned to a machine, starting at the point where the machine breakdown start. All subsequence orders are scheduled later in time, by an amount that roughly works out to be the duration of the breakdown period. In some cases, a more sophisticated approach is required where a complete re-optimization is performed to repair damage that was done to the percentage of orders that would be delivered on time. This kind of optimization must attempt to keep

as much of the existing schedule intact while repairing the placement of the affected orders.

**Freeze periods:** Every time the EA is run, the possibility exists that a very different schedule could be produced when compared to the previously generated one. This happens simply as a result of the EA doing its job of finding an optimal solution as dictated by its objective function (which changes each time new orders and constraints appear in the system, which is virtually always). In a real-world application, it is highly unlikely that this kind of approach could be tolerated. The preferred mode of operation is the following. The software is used to initially create a schedule for a fairly long period of time, for example 2-4 months. However, once that schedule is accepted and saved to an internal database, subsequent daily use of the optimizer to schedule newly arriving customer orders must be done in a controlled manner so that there is a buffer period at the beginning of the schedule that remains the same as if it was yesterday. This unchanging portion of the schedule is referred to as a *freeze period*. It is usually defined in terms of a number of days, for example 7 days. In order to achieve this effect, the EA has to perform what we refer to as seamless *meshing* with an existing schedule. The first 7 days of the existing schedule are frozen, and some of the previously existing orders, along with new ones are re-shuffled by the optimization algorithm and attached in a neat continuous way with the orders in the freeze period. The contents of the freeze period vary with time, as each day some orders are executed and so are removed completely from the schedule, and others that were once outside the freeze period gradually move into it. This is referred to as a *rolling time window*.

On a more granular level, freeze periods are affected by live update feedback data coming from the factory floor. The *current order* on a given bottling line is the first order showing up on the schedule. At any given moment, it usually corresponds to an order that physically is in the process of being executed in the factory. Our system was designed to receive updates in near-real-time (every 5 minutes) letting it know how much of that order has been carried out. That in turn affects the orders in the freeze period because the size of the current order needs to be gradually reduced to reflect what remains to be done, and consequently at the right-hand end of the freeze period, more orders will come in. This sometimes leads to a situation in which there is an order that straddles the freeze period and the open-optimization period.

Figure 7 shows a screen capture from our software application that illustrates how the frozen period is managed. In the graphical area of the window, where orders are represented by rectangles, on the left-hand side of the screen there are a number of orders that are surrounded by a thick bold border, indicating that they have been frozen. On the right-hand side, are the orders that the optimizer is free to move around to find an optimal schedule.

**Modified orders:** Once a schedule has been created and saved, there might be a situation in which the next time the software package is opened, it

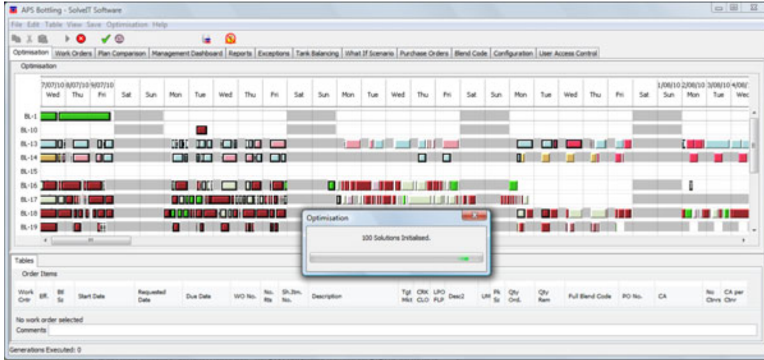


**Fig. 7.** Freeze Period Illustration

realizes that there was a modification made to one of the scheduled orders in the database. This might be, for example, a change in quantity. More or fewer bottles of wine may be required and by adjusting the scheduled order, the start and end times of all subsequent orders on the same bottling line become affected.

**Poor/Excellent Job Execution:** From historical performance data, any given bottling machine has an expected production rate. This rate is used by the software to estimate the time required to run a job of a particular size. However, due to a number of variables, the efficiency of a bottling machine may be better or worse than the expected rate on any given day, and the factory manager would expect the scheduling optimizer to take this into account when creating a new schedule, or when adjusting an existing one. A machine that is performing poorly will delay jobs scheduled further down in time, and likewise a machine that is performing unusually well, will bring jobs up earlier, which could, in some cases, have a detrimental effects when we take into consideration the availability of dry goods. The optimization software must be able to deal with this time-varying issue by re-adjusting the solution, or by re-optimizing as necessary.

## 6 The Solution Using an EA

In this section, we will look at the structural, algorithmic, and programmatic details required to solve the problem using an EA. We begin by examining the representation of a candidate solution and then look at how that abstract representation is converted (decoded) into an actual schedule with dates and times. Next we look at the key operators that were employed to alter candidate individuals. Finally we re-visit the time-varying problems described above and show how various elements of the EA had to be modified to accommodate those issues.

### 6.1 Representation

One of the most important aspects of the solution of a problem using an EA is the representation used to encode a candidate solution. Forcing the use of a particular representation may significantly impact on the quality of the solutions found since many useful operators may be overlooked, or may be cumbersome to program, thereby slowing down the execution of the algorithm. For this application, we chose a representation that closely matches the actual assignment of orders to machines. For reasons that will be described below, we chose to use both a genotypic and phenotypic representation of a solution.

For the wine bottling scheduling problem, the core of the problem was conceptualized as having a number of orders that must be placed on a fixed number of bottling machines in an efficient sequence. Hence the natural representation to use is a mapping of lists of orders to machines. This can be visualized as in Figure 8. The representation illustrated in this diagram is quite similar to the final schedule presented visually in Figure 3 above. The difference is that the actual decoding of the individual into a real-world schedule also takes into consideration several other time-varying factors such as machine availability, splitting of single orders into several sub-jobs, meshing with an existing schedule, and so on.

The representation shown in Figure 8 is stored programmatically as a map of machines to variable-length lists of orders. The list for any given machine is sorted chronologically in terms of which orders will be carried out first. The individual is constructed in such a way that the assignment of orders to machines is always valid, in other words, the assigned orders always respect product routings.

From a formalised perspective, the search space can be thought of as the set

$$S = ((m, o))^n \tag{1}$$

$m \in M$, where $M$ is the set of bottling machines, and $o \in O$, where $O$ is the set of customer orders to be scheduled, and $n$ is the number of such orders, i.e., $n = |O|$. Hence an individual (as illustrated in Figure 8) can be represented mathematically as

$$I \subseteq S \tag{2}$$

If we wanted to use a mathematical structure that emphasizes the ease of accessing the jobs assigned to each machine, then instead, we could think of each individual as being represented as

$$I = (a_1, a_2, \ldots, a_k) \tag{3}$$

where, $k$ is the number of bottling machines, and $a_i = (o_{i1}, o_{i2}, \ldots, o_{i\alpha(i)})$ is a sequence of customer orders assigned to machine $i$, and $\alpha(i)$ is the number of orders assigned to that machine.

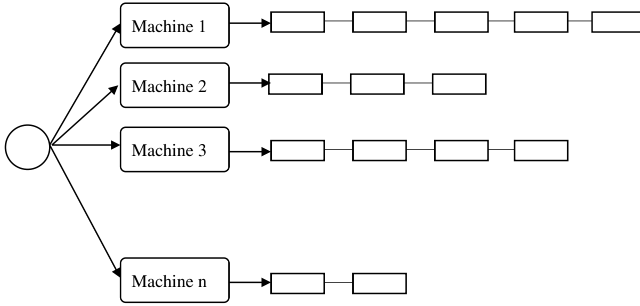


**Fig. 8.** Scheduling Individual Representation

### *6.2 Decoding*

In order to manage the decoding process, we employed a concept we referred to as *time blocks*. A time block, illustrated in Figure 9 is a data structure that keeps track of a contiguous period of time. Each such block keeps track of a start time, an end time and an activity that is performed during that time. The time block data structure allows for the possibility that nothing is actually done during the period of time, in which case the time block is referred to as an *available* time block. The link to the activity performed during a time block may point to an external data structure that contains any information on any level of detail required to accurately model a scenario.

Decoding begins with a series of multiply-linked-lists of time block nodes associated with each machine. Each machine has a list of available time blocks, and occupied time blocks. At the outset, before anything is placed on a

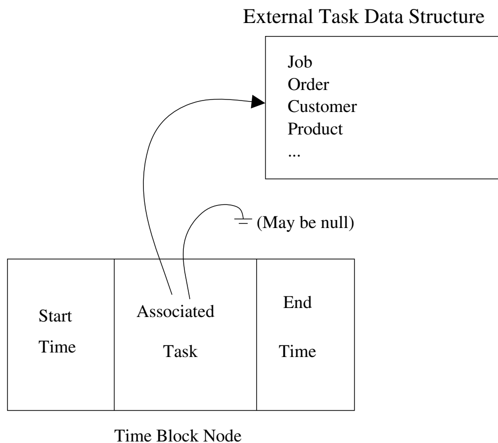


**Fig. 9.** A Time Block Node

machine, it would only contain a list of available time blocks, each representing a chunk of time during which the machine is available for use. This is what would happen in the nominal scenario. If there are manual decisions that were already made by a human operator, then these would be reflected by the existence of some occupied time blocks. More detail on this issue is given in section 6.5 below.

Decoding proceeds by going through all machine-job pairs found in the individual representation and proceeding to the corresponding machine, finding an available time block node, and marking it as occupied for the corresponding job. Some jobs may not fit into the first available time block and may need to be split into multiple parts. How this is done, and if it is permissible at all, depends on the policies of the business for which the scheduling application is being created. In the case of the wine bottling application that we are considering, orders were split across adjacent available time blocks. This process is illustrated in Figure 10.

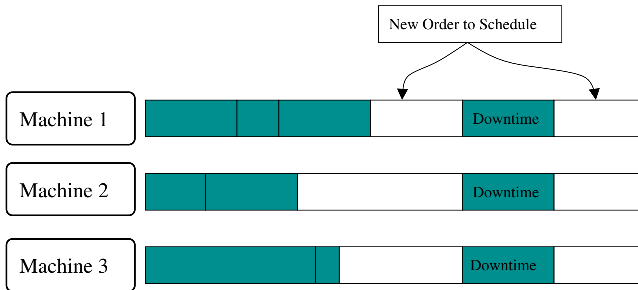


**Fig. 10.** Illustration of the Decoding Process

An important issue that arises when decoding is performed in this way for a scheduling problem is the question of in what order should the machine-job pairs be considered. Should we select Machine-A first and process all of the orders in its list, and then proceed to Machine-B and so on? Alternatively, should we process one job from Machine-A's list, then one job from Machine-B's list and so on and eventually cycle back to Machine-A, and keep on in that manner until all jobs have been decoded? The answer to this question depends on the industry being considered. For the wine bottling problem there is no dependency or constraints between jobs running on different machines and hence, the former approach was used.

However, there are cases in which the second approach (or yet others) may be needed. For example in some industries there might be a limit to the number of changeovers that are permitted to occur across an entire plant in one day. In those cases, it is important not to bias the decoding (or alternatively to deliberately and carefully bias it) to favor one machine, or a group of machines.

### 6.3 Operators

A number of operators were used to manipulate the representation given above. To avoid the problem of having to perform extensive repairs based on invalid representation states, crossover-type operators were avoided. The operators listed below, which are typical examples from the set used, may all be considered as mutation operators.

- *Routing Mutation*: This operator modifies the machine that was selected to execute a job. An alternative is randomly chosen from the set of possible options.
- *Load Balancing Mutation*: This is a variation to the routing mutation operator which, instead of merely randomly choosing an alternative machine for an order, chooses from a subset of machines that are under-loaded. Such an approach would help with load-balancing of the machines.
- *Grouping*: This operator groups orders based on some common characteristic, such as wine color, bottle type, or destination export country. There are several variants of the grouping operators. Some operate quite randomly, looking for a group of orders based on some characteristic and then looking left or right for a similar group and then merging the two. Other examples are given below.
- *Recursive Grouping*: A more directed variation on grouping is recursive grouping. This operator seeks out an existing group of jobs based on wine color, then within that group performs random grouping based on some other characteristic, such as bottle size. This process may then be repeated inside one of the subgroups.
- *Outward Grouping*: Outward grouping is a term we use to describe the process of identifying groups in a list of jobs based on a primary characteristic, then randomly selecting one of them and from that location looking left and right for another group with a common secondary characteristic and finally bringing together the two. As a concrete example, we may first identify groups based on a primary attribute, say closure type, that is, whether a screw cap or a cork is used to close the bottle, and then randomly select a group that uses screw caps as a starting point. Next a secondary attribute such as wine color is looked at. Suppose the group of screw caps involves white wines, then we look randomly to the left or to the right for a group based on the primary attribute that also involved white wines. In the end we may end up merging with a group of jobs that involves corks, but which happens to be all white wines. The end result is that we have a larger continuous group of white wine, with two sub-groups, one with screw caps and the other with corks. The process of finding a starting point and then looking outwards for subgroups to merge with gave rise to the name *outward grouping*.
- *Order Prioritization*: Orders that may be showing up as being produced late after decoding an individual are stochastically prioritized by moving them left in the decoding queue. This type of operator could be made more

intelligent by moving groups of jobs along with the one that is identified as being late. That way the job gets prioritized, but at the same time disruption to grouping is minimized. This operator works on one of the most important objectives of the solution, to maximize DIFOT.

## 6.4 A Pseudocode View of the EA

The coding and implementation of the algorithms used to solve the problem of this chapter are very lengthy, but from the point of view of core principles, the key parts of the EA can be succinctly summarized as in Algorithms 1, 2 and 3.

**Algorithm 1:** Main Loop of Algorithm

```
1 initialize()
2 while numIterations < targetIterationCount do
3     individual = selectRandomIndividual()
4     operator = rouletteSelectionOfOperator()
5     newIndividual = operator.execute(individual)
6     if evaluation(newIndividual) > evaluation(individual) then
7         individual = newIndividual;
```

**Algorithm 2:** Population Initialization

```
1 for i = 1 ... populationSize do
2     individual[i] = newBlankIndividual()
3     for each order in customerOrdersList do
4         capableMachine = order.getCapableMachines()
5         selectedMachine = uniformRandomSelection(capableMachines)
6         individual[i].assign(order, selectedMachine);
```

**Algorithm 3:** Individual Evaluation

```
1 evaluation = 0
2 evaluation := evaluation - delayedOrdersPenalty()
3 evaluation := evaluation - colourChangeoverTimePenalty()
4 evaluation := evaluation - toolChangeoverTimePenalty()
5 return evaluation
```

## 6.5 Solving the Dynamic Issues

We will now look at how we dealt with the time-varying issues that were identified in Section 5. As will be observed, the problems were addressed by a

combination of modifications made to the initial time block node linked-lists, to the decoding process, by altering the input variables to the optimizer, and by introducing a step between the optimizer and the human user, called *solution re-alignment*.

**Solving Manual Overrides:** This problem was solved by applying a constraint to the decoding process, and indirectly affecting the fitness function. The software application allows the user to select a particular order, and specify which machine it should be done on, as well as the date and time of assignment. This becomes a timetable constraint for the genotype decoder. When a candidate individual is being decoded, the initial state of the machine time block usages includes the manually assigned orders as part of the list of occupied time blocks. Subsequent decoding of non-manually-assigned jobs would take place as usual using the remaining available time blocks.

The fitness function would then evaluate how well the individual was decoded into the partially occupied initial machine state. As in the case where there are no manual assignments, it is up to the evolutionary operators to modify the individual in such a way that its decoded form meshes well with the fixed orders to reduce changeovers and so on. The initial state of available time blocks used for decoding is illustrated in Figure 11.

With this approach, it is primarily the fitness function that would guide the search to a relatively good solution built around the manual assignment. However it is also important to de-emphasize some of the more structured and aggressive operators such as recursive grouping and outward grouping, which were designed with a clean slate in mind. They would still contribute toward the search for a good solution in parts of the time period that do not contain manual assignments, but the main EA loop should keep track of the performance of these operators and adjust their probability of application accordingly.

**Solving Freeze Periods:** The application allows the user to select a freeze date and time on each machine used in the bottling plant. During an optimization run, all assignments in the existing schedule that are before the freeze period cut-off are internally marked as manual assignments, and therefore behave in exactly the same way as described above for user-defined manual assignments.

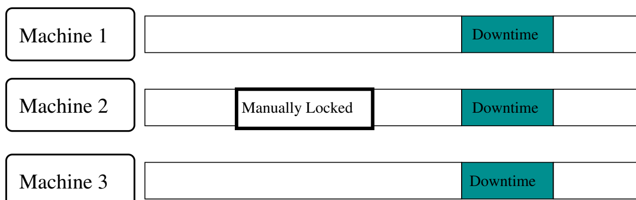


**Fig. 11.** Modified Initial Time Blocks for Manual Locks

It is also important to pay attention to jobs that might straddle the freeze period cut-off date. Such orders need to be fully frozen to ensure correct results. Another situation that needs attention is where a job may be split into several pieces, for example due to relatively low machine availability on consecutive days, and one some of those split pieces fall into the frozen period. Care must be taken to also freeze the parts that are outside of the freeze period, but which belong to the same order.

**Solving Machine Breakdowns, Modified Orders, and Poor / Excellent Job Execution:** These three problems were solved using a similar approach. Take Modified Orders for example. When the application re-loads and realizes that an existing order has been modified, for example its quantity has been increased or decreased, then the necessary action to be taken is at the level of modifying the existing solution, prior to it being fed back into the next run of the EA. This was accomplished using a process called *solution re-alignment*.

Essentially, on each machine, all assigned orders, sorted by starting date, are examined for any changes to the underlying orders. A new instance of an available time block nodes linked-list is created and the assigned orders are then re-inserted into it. By comparing each order as it last existed in the schedule with the order record coming from the company's ERP system allows assignments being shortened or lengthened as needed. Following the re-insertion of a modified assignment, all subsequent assignments are modified to fit in the resulting changed available space. This process has a direct effect on the number of orders ending up in the freeze period and therefore on the amount of available time that the optimizer has at its disposal for the next run. The process of solution re-alignment is illustrated in Figure 12.

Similarly, if unexpected machine breakdowns were encountered, then the linked list of available time block nodes into which the existing schedule is re-aligned is adjusted to reflect the new pattern of available operating times, with the breakdown periods marked as unavailable.

Solution re-alignment is a step that links the internal decoded representation of an individual with the dynamic world of the human operator in which multiple constraints can vary from one use of the application to another. It allows an existing solution to be adapted to match constraints that may have

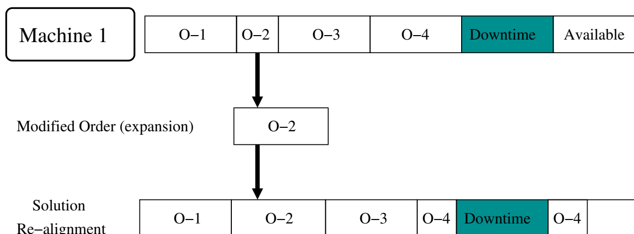


**Fig. 12.** Realigning a Solution

changed, and possibly subsequently, allow the EA to run with a freeze period that now accurately reflects the effect of the changed constraints. Re-alignment can be thought of as a second-level decoding process that works with an already decoded representation as opposed to working with a genotypic representation.

## 7 Conclusions

In this chapter, we have looked at issues that arise when dealing with dynamic optimization problems in real-world applications. A literature review was presented in line with our classification of dynamic problems according to time-varying objective functions, input variables and constraints. An important consideration that was emphasized throughout the chapter was that complexities arising due to practical, time-varying constraints had to be dealt with adequately in order to ensure that the solution produced is usable in an actual business environment. Given that a large real-world software application was presented as a main outcome of the research presented here, the need to ensure that those issues were resolved in a practical way that still allowed the power of EAs to be applied was a main theme.

The issues raised were exemplified by a case study based on an EA scheduling optimizer that was implemented for use in the wine bottling industry. The software, having been deployed and put into daily use in a live production environment, serves as empirical evidence that the approaches put forward in this chapter have passed the litmus test of suitability for real-world applications.

Numerical experiments were not included as part of this chapter, for the simple reason that they would be almost meaningless, given the objective of the research, which was not to compare one EA method against any other, but rather to demonstrate a way of marrying an EA with supporting software structures to enable the final software application to deal with difficult, practical day-to-day business realities.

One of the key issues that were dealt with was rooted in the need to be able to allow a human operator to manually decide certain parts of the eventual solution. In effect what this does is to modify the search space explored by the EA. Although there are undoubtedly better solutions that exist in the unconstrained search space, when we take into account these so-called manual assignments, the optimal solution based on those restrictions must be found in order to satisfy the immediate business needs that gave rise to the need for human intervention into the search process.

Another key issue that was explored was the need to ensure that future schedules mesh seamlessly with existing ones. This was handled by extending the concept of manual-assignments to the concept of a freeze period in which existing scheduled assignments are treated as unchanging, and therefore tantamount to numerous manual assignments. This has the benefit of allowing factory operators to continue working with the assurance that preparations

made for the next few upcoming days would not be disrupted, and planning could proceed in a smooth and normal manner.

Overall, what has been illustrated is that in implementing a scheduling EA for use in a practical commercial application, it is necessary to design almost everything, from the representation, the decoding process, the operators and the solution structure, in such a way that maximum flexibility is maintained with respect to allowing time-varying constraints to be easily considered by the core algorithm that finds an optimal schedule, and with respect to allowing an easy and natural flow between data structures used directly by the optimization algorithm and the user interface that is manipulated by the human operator.

**Acknowledgements.** This work was partially funded by the ARC Discovery Grant DP0985723 and by grant N516 384734 from the Polish Ministry of Science and Higher Education (MNiSW).

## References

1. Davis, L.: Job shop scheduling with genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 136–140. L. Erlbaum Associates Inc., Mahwah (1985)
2. Van Laarhoven, P.J.M.: Job shop scheduling by simulated annealing. Operations Research 40(1), 113–125 (1992)
3. Wang, L., Zheng, D.-Z.: A modified genetic algorithm for job-shop scheduling. International Journal of Advanced Manufacturing Technology 20(1), 72–76 (2002)
4. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1(2), 117–129 (1976)
5. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proc. of the 1999 Congress on Evolutionary Computation, CEC 1999, pp. 1875–1882 (1999)
6. Branke, J.: Evolutionary approaches to dynamic optimization problems - a survey. In: GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pp. 134–137 (1999)
7. Morrison, R.W., De Jong, K.A.: A test problem generator for non-stationary environments. In: Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999, pp. 2047–2053 (1999)
8. Carlisle, A., Dozier, G.: Adapting particle swarm optimization to dynamic environments. In: International Conference on Artificial Intelligence, Las Vegas, NV, USA, pp. 429–434 (2000)
9. Yu, X., Tang, K., Yao, X.: An immigrants scheme based on environmental information for genetic algorithms in changing environments. In: Proceedings of the 2008 Congress on Evolutionary Computation, CEC 2008, pp. 1141–1147 (2008)
10. Pettit, E., Swigger, K.M.: An analysis of genetic-based pattern tracking and cognitive-based component tracking models of adaptation. In: Proceedings of the National Conference on Artificial Intelligence, pp. 327–332. AAAI Press, Menlo Park (1983)

11. Krishnakumar, K.: Micro-genetic algorithms for stationary and non-stationary function optimization. In: Proc. of the SPIE, Intelligent Control and Adaptive Systems, pp. 289–296 (1989)
12. John, J.: Grefenstette. Genetic algorithms for changing environments. In: Parallel Problem Solving from Nature, vol. 2, pp. 137–144. Elsevier, Amsterdam (1992)
13. Branke, J.: The moving peaks benchmark, `http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/movpeaks/`
14. Johnston, M.D., Adorf, H.-M.: Scheduling with neural networks – the case of the hubble space telescope. Computers & Operations Research 19(3-4), 209–240 (1992)
15. Chryssolouris, G., Subramaniam, V.: Dynamic scheduling of manufacturing job shops using genetic algorithms. Journal of Intelligent Manufacturing 12(3), 281–293 (2001)
16. Madureira, A.M., Ramos, C., Silva, S.C.: Madureira, Carlos Ramos, and Slvio C. Silva. Using genetic algorithms for dynamic scheduling. In: 14th Annual Production and Operations Management Society Conference, POMS 2003 (2003)
17. Emperador, J.M., González, B., Winter, G., Galván, B.: Minimum-cost planning of the multimodal transport of pipes with evolutionary computation. Int. J. Simul. Multidisci. Des. Optim. 3(3), 401–405 (2009)
18. Jain, A.K., Elmaraghy, H.A.: Production scheduling/rescheduling in flexible manufacturing. International Journal of Production Research 35(1), 281–309 (1997)
19. Petrovic, D., Alejandra, D.: A fuzzy logic based production scheduling/rescheduling in the presence of uncertain disruptions. Fuzzy sets and systems 157(16), 2273–2285 (2006)
20. Kutanoglu, E., Sabuncuoglu, I.: Routing-based reactive scheduling policies for machine failures in dynamic job shops. International Journal of Production Research 39(14), 3141–3158 (2001)
21. Holthaus, O.: Scheduling in job shops with machine breakdowns: an experimental study. Computers & Industrial Engineering 36(1), 137–162 (1999)
22. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms—i: representation. Computers & Industrial Engineering 30(4), 983–997 (1996)
23. Yamada, T., Reeves, C.R.: Solving the $c_{sum}$ permutation flowshop scheduling problem by genetic local search. In: Proceedings of the 1998 Congress on Evolutionary Computation, CEC 1998, pp. 230–234 (1998)
24. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research 91(1), 160–175 (1996)
25. Marchiori, E., Steenbeek, A.: An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In: Oates, M.J., Lanzi, P.L., Li, Y., Cagnoni, S., Corne, D.W., Fogarty, T.C., Poli, R., Smith, G.D. (eds.) EvoIASP 2000, EvoWorkshops 2000, EvoFlight 2000, EvoSCONDI 2000, EvoSTIM 2000, EvoTEL 2000, and EvoROB/EvoRobot 2000. LNCS, vol. 1803, pp. 367–381. Springer, Heidelberg (2000)
26. Ponnambalam, S.G., Mohan Reddy, M.: A ga-sa multiobjective hybrid search algorithm for integrating lot sizing and sequencing in flow-line scheduling. International Journal of Advanced Manufacturing Technology 21(2), 126–137 (2003)

27. Ray, T., Sarker, R.A.: Optimum oil production planning using an evolutionary approach. In: Evolutionary Scheduling, pp. 273–292. Springer, Heidelberg (2007)
28. Burke, E.K., Smith, A.J.: A memetic algorithm to schedule planned maintenance for the national grid. Journal of Experimental Algorithmics 4, 1 (1999)
29. Martinelli, F.: Stochastic comparison algorithm for discrete optimization with estimation of time-varying objective functions. Journal of Optimization Theory and Applications 103(1), 137–159 (1999)
30. Tinós, R., Yang, S.: Genetic algorithms with self-organizing behaviour in dynamic environments. In: Evolutionary Computation in Dynamic and Uncertain Environments, pp. 105–127. Springer, Heidelberg (2007)
31. Yang, S.: Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Evolutionary Computation in Dynamic and Uncertain Environments, pp. 3–28. Springer, Heidelberg (2007)
32. Schönemann, L.: Evolution strategies in dynamic environments. Evolutionary Computation in Dynamic and Uncertain Environments, pp. 51–77. Springer, Heidelberg (2007)