Time-Varying Constraints and Other Practical Problems in Real-World Scheduling Applications

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

***Abstract*— When an evolutionary algorithm is used as an optimizer in a scheduling software application that is destined for use in a real-world commercial setting, a number of time-variability issues are encountered. This paper explores several such issues and other practical problems that arose during the solution of a scheduling application in the area of wine bottling. Each hurdle was addressed by appropriately adjusting the candidate individual representation, the procedure used to decode an individual, or the objective function itself. Addressing these issues is critical when designing and constructing the evolutionary algorithm, in order to ensure that the resulting system is robust enough to meet the demands of day-to-day use. The approach described in this paper has been proven by implementation and vigorous sustained use in a complex business environment.**

I. Introduction

Scheduling problems have long been tackled by evolutionary algorithms [1], [2], [3]. The general problem is one of assigning a number of tasks to a limited set of available machines in such a way as to minimise the amount of time required to complete all of the tasks. In its various forms, this is equivalent to the NP-complete problem of job shop scheduling [4], hence making it a prime candidate for approximate solution using evolutionary algorithms. In brief, a candidate solution can be represented by an ordered list of (machine, job) pairs indicating a natural sequence of execution. And, at a minimum, a simple mutation operator can affect this representation, with an appropriate fitness function, reflective of the various problem constraints and parameters, guiding the population to a good solution.

However, when an attempt is made to use this approach to create a fully-fledged application that can be used in a real-world business environment to manage the day-to-day scheduling needs of a large enterprise, one is faced with a number of issues that require careful consideration in order to reach a satisfactory resolution. The business needs of the area of application must be met in a way that is compatible with the architecture of the evolutionary algorithm employed as the core element of the overall solution.

Arvind Mohais (am@solveitsoftware.com), Sven Schellenberg (ss@solveitsoftware.com) and Neal Wagner (nw@solveitsoftware.com) are with SolveIT Software Pty Ltd, 99 Frome Street, Adelaide, SA 5000, Australia.

Maksud Ibrahimov (maksud.ibrahimov@adelaide.edu.au) is with the School of Computer Science, University of Adelaide, SA 5005, Australia.

Zbigniew Michalewicz is with the School of Computer Science, University of Adelaide, South Australia 5005, Australia. Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland, Polish-Japanese Institute of Information Technology, ul. Koszykowa 86, 02-008 Warsaw, Poland (email: zbigniew.michalewicz@adelaide.edu.au)

Some examples of key business application issues that arise in the wine bottling industry, and which are also quite typical in many other areas of application, are, the need to be able to specify manual decisions for certain parts of the schedule, and the need for continuity with previously published production schedules. The approach used to solve these, and other issues, will be presented, and the reader should find that the structure of the solutions employed are extensible to scheduling applications in general.

This paper begins, after a brief literature review, by giving a general description of various categories of time-varying issues that arise in dynamic optimization problems. This is followed by a full description of a real-world dynamic scheduling problem, namely the problem of scheduling production orders for a wine bottling plant. This problem is used as a case study to illustrate the ideas presented. The business issues that were necessary to take into consideration while building an EA-based solution to the problem are then described in detail, and finally, a description of the evolutionary algorithm used as the final solution is given with emphasis on how the various issues were addressed.

II. Scheduling Problems in the Literature

In this section, we take a brief look at some papers that give the reader a cross-section of examples of the application of evolutionary algorithms to scheduling applications, both classic academic problems, as well as real-world problems, as well as examples of using EAs for dynamic optimization problems (which are discussed further in the next section).

Job shop scheduling problem is a canonical example of a scheduling problem. It is one of the most difficult combinatorial optimisation 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 [5]. [1] discusses the application of genetic algorithms to the job-shop scheduling problem. Wang and Zheng solve it using a modified genetic algorithm [3]. In their work, to avoid premature convergence and improve neighborhood search, the classical mutation operator was replaced by the metropolis sample process of simulated annealing with a probabilistic jumping property. A special crossover operator was used in the simulate annealing-based main loop. In [2] simulated annealing approach to tackle job-shop scheduling problem is discussed.

Yamada and Reeves [6] proposed an evolutionary algorithm to solve the permutation flow shop problem. They combined stochastic sampling and best descent methods into one unified method to reach effective results. They used

978-1-4244-8126-2/10/$26.00

multi-step crossover and mutation as well as a representative neighborhood method, which is constructed by clustering the original neighborhood and choosing the best representative from each cluster. This method was described by Nowicki and Smutnicki in their taboo search algorithm for the flow shop problem [7].

There are also many published reports of the application of evolutionary algorithm to solve scheduling problems in various domains of application.

Marchiori and Steenbeek in [8] developed an evolutionary algorithm 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 [9] 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 [10]. 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 limited amount of gas each day. Single objective and multiple objective versions of the problems were considered.

Burke and Smith [11] 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. A similar study was carried out by [12].

Martinelli in [13] shows the optimisation of time-varying objective functions using stochastic comparison algorithm. Values of the time-varying functions are known through estimates. Noise filtering was introduced to decrease probability of wrong moves.

Tinós in [14] introduces self-organizing random immigrants scheme for algorithms in dynamic environments. Newly immigrated individuals are held in the subpopulation for some time until they develop good fitness values.

Yang in his chapter [15] discusses memory scheme approach as a method of improving performance of evolutionary algorithms for dynamic environments. Two kinds of memory schemes described: direct and associative. These schemes are applied to genetic algorithms and on univariate marginal distribution algorithms in dynamic environments. These algorithms were run on a set of generated tests of: cyclic, random and with addition of noise.

Lutz in his work[16] considers the application of evolutionary strategies for numerical optimisation problems in dynamic environment. Main parameters of evolutionary strategies for problems in dynamic environments are presented. Performance measures discussed with advantages and disadvantages of each of them.

III. Dynamic Optimization Problems

In this section we look at some general characteristics of real-world scheduling problems. It is well-known that these problems are highly dynamic in nature, and there is a considerable body of EA research literature to address this fundamental property of such optimization problems. Dynamic problems may be categorized into three groups:

1) *Time-varying objective functions*. These are problems in which the shape of the fitness landscape varies with time. The location of the optimum value is continually shifting. There has been a considerable amount of research done in this area, for example [17]. In many published reports, situations were set up wherein during a single run of an evolutionary algorithm, 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. In the case study that will be considered next in this paper, we have encountered a time-varying objective function, but 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) *Time-varying input variables*. These are problems in which the input data being processed in the optimization scenario change 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 we are dealing with a live business environment, and as time passes, old orders are completed, and new ones arrive.
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. An example of a varying constraint is the requirement that a new production schedule be created that meshes seamlessly with the existing schedule once a number of days of the current production schedule have been fixed. 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 creates a new solution that matches up with the existing one, and still produces an optimal result for the long term.

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 are those that belong to the second and third categories. Interestingly we have found that the vast majority of pub-

lished research in evolutionary algorithms 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 are effectively ignored. Figure 1 illustrates some examples of issues that typically arise in real-world problems.

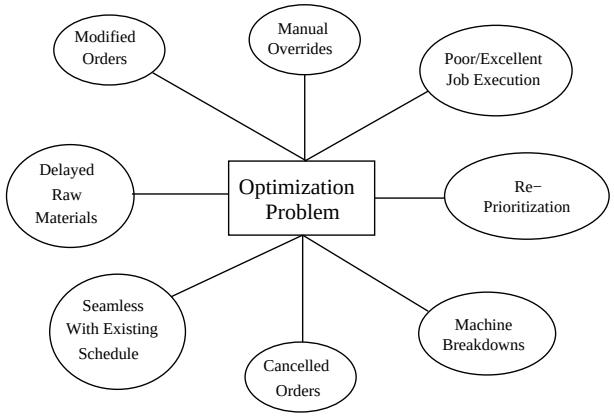

Fig. 1. Business Issues Affecting Scheduling

IV. Scheduling Case-Study: Wine Bottling

The Wine Industry and Evolutionary Algorithms are a particularly good combination. The former is the source of a host of real-world application areas that provide formidable challenges to the latter. In this section, we explore one aspect of the production of wine, namely the packaging of the product, or more specifically the bottling of the bulk consumable liquid.

In essence, the problem context can be thought of as having a number customer orders for particular finished goods, each of which must be completed by a given due date. A packaging factory has multiple bottling machines available for use, each capable of doing a subset of the types of finished wine products manufactured by the company. The problem is to determine a sequence of orders to be performed on each bottling machine in such a way as to produce all orders by their due dates, and at the same time minimise production costs.

Due to the highly variable and complex nature of the wine making business, what would otherwise be considered a straightforward scheduling application for an evolutionary algorithm, becomes a significant challenge. These practical considerations that make this a highly dynamic problem of the third kind.

The problem of bottling wine is primarily a scheduling problem, but becomes much more than that when we take into consideration the intricacies involved in running a real-world dynamic business. Large wine manufacturing companies produce wine in bulk, fermenting, filtering, stabilizing and eventually storing finished liquids in enormous tanks ranging in size from 50,000 liters to 1,000,000 liters or more. The wine remains in these vessels until such time as there is a demand to have it packaged into bottles for final shipment to customers. The liquid is pumped from the large storage tanks into smaller intermediate tanks that are used to feed machines that then fill bottles, cork them, label them, and so on. In most cases, demand is known ahead of time because customers place orders with sufficient notice. These orders must then be examined to determine when, and in what order they should be fulfilled. This is the basic part of the wine bottling problem, i.e. determining how to use the limited number of bottling machines in the most efficient and cost-effective manner possible.

Some typical scheduling issues faced are:

- *Due dates*: Orders have dates by which they must be completed. This of course is one of the most important constraints as we do not want to displease customers who are expecting their products to be delivered on time.
- *Availability of the bulk wine*: Jobs can only be scheduled when the needed liquid wine is ready in the bulk storage tanks. Some batches may still be undergoing processing such as fermentation, filtration, and temperature stabilization.
- *Availability of dry goods*: A finished bottle of wine is put together from several dry good components. For example, the glass bottle itself, the screw cap or cork, several labels, and other such items. An order for a certain number of bottles of a particular type of wine, therefore requires corresponding amounts of each dry good. These items are stocked at the a warehouse facility and the optimizing software must be able to make scheduling decisions based on the whether or not the appropriate amount of dry goods are available at a particular time.
- *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.
- *Colour changeovers*: There are two basic types of wine, red and white, and several other variations such as rose, sparkling red, sparkling white, fortified and so on. A machine that is in the process of bottling one type of wine and must stop and change over to another type, needs to undergo some cleaning. The extent of the cleaning depends on the type of change. For example a machine that is currently bottling white wine and must switch over to red can go through q relatively quick cleaning process because if a small amount of white wine enters into the next product which is red, there would not be any real damage done. On the other hand, if we are changing over from red wine to white wine, then quite the opposite is true. There needs to be an extremely thorough cleaning and sterilization process. Not a drop of red wine can be allowed to get into a white product.
- *Other changeovers*: There are other considerations that affect machine setup time, such as what kind of cap or cork is being used on the wine bottle, the type and size

of bottle, the number of labels being used, and number of bottles to be put into boxes and so on.

- *Machine availability*: Machines have normal hours of operations, which may be for example 8:00am to 6:00pm. During certain periods of the year, these times may vary. Furthermore, sometimes machines are taken down for repairs and servicing. The optimizer must take this into consideration so as to not make incorrect scheduling assignments.
- *Routings*: Each finished product (also sometimes referred to as a Stock Keeping Unit (SKU)) has a certain set of machines on which it can be produced. On each different machine on which it can be produced, there might be different attributes related to its production. For example, the speed at which it can be produced on machine A might be faster than on machine B. Each information record related to the possibility of producing an SKU on a particular machine is referred to as a *routing*, and the optimizer uses this to determine where to place orders.

The end result of our efforts to develop an evolutionary algorithm that could deal with all of these issues and create a feasible, optimal schedule, was a full-featured piece of software that was deployed for a major global wine company, and is currently in daily use at wine bottling sites internationally. The planning period varies according to demand from two months to about four months, and the number of orders being scheduled can vary from 400 to 1000 (typical numbers). Figure 2 shows a screen capture from this application, with all confidential client information removed. The graphical area in the upper part of the window shows an intuitive visual representation of what the production schedule looks like. Each row of rectangles represents a particular bottling machine, and each of the rectangles represents an order that has been assigned to the corresponding machine. the length of the rectangle is proportional to the run time of the job on that machine.

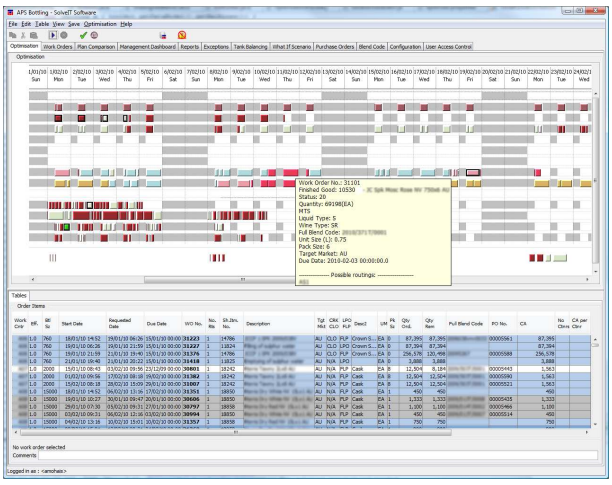

Fig. 2. Software Screen Capture

V. Time-Varying Challenges in Wine Bottling

- *Manual overrides*: There are times when the human scheduler in charge of running the bottling plant needs to make decisions that override what the software application has determined is the best sequence in which to run a set of jobs. One reason for this might be that a very important customer has made a late, but urgent request for a quantity of wine that would normally be scheduled for production several weeks into the future. However, due to the higher priority accorded to this customer, the wine company will deliberately allow an inefficient sequence of jobs to run for a temporary period of time, in order to satisfy this order. The software application that we developed had to be flexible enough to allow its built-in evolutionary algorithm to work in such a way that it can 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. In other words, it must build an optimal solution around certain fixed sub-optimal constraints.
- *Machine breakdowns*: From time to time, a bottling machine 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. In some cases this may be as simple as shifting a sequence, but in other situations it could mean that a complete re-optimization is required.
- *Freeze periods*: Each time the evolutionary algorithm is run, it could potentially radically move around items in the previous existing schedule, because by doing so it finds a solution that is much better when considering new orders that might have been loaded into the system. However, in practice this cannot be allowed. Once a schedule has been set for a period of time, say 3 months, at least some of it must remain constant. The factory managers and workers would have looked at the next 2 weeks say, and already started moving around hardware and other resource in order to prepare for the coming work. It is unacceptable to come along in a day or two and tell them that the plan has changed. The period of time at the start of a schedule for which nothing must change is referred to as a ‘freeze’ period. The optimizer must be able to continue optimizing outside of this freeze period, and at the same time mesh seamlessly with what was frozen during this time.
- *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 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 less 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*: Due to a number of variables, the efficiency of a bottling machine may be better or worse 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.

VI. The Solution Using an Evolutionary Algorithm.

A. Representation

One of the most important aspects to the solution of a problem using an evolutionary algorithm 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 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 one of a mapping of lists of work orders to machines. This can be visualized as in Figure 3. The representation illustrated in this diagram is quite similar to the final schedule presented visually in Figure 2 above. The difference is that the actual decoding on the individual into a real-world schedule also takes into consideration several other time-vary factors such as machine availability, splitting of single orders into several sub-jobs, meshing with an existing schedule and so on.

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

$$S = ((m, o))^n$$

$m \in M$, 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 3 can be represented mathematically as

$$I \subseteq S$$

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

$$I = (a_1, a_2, \ldots, a_k)$$

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.

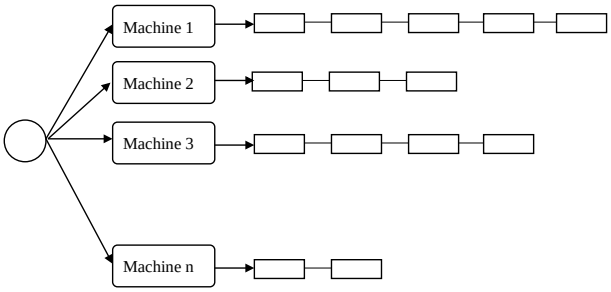

Fig. 3. Scheduling Individual Representation

B. Decoding

Decoding begins with a series of multiply-linked-lists of so-called 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 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 VI-D below.

Decoding involves going through the 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 be able to fit in the first available time block, and may need to be split into multiple parts. How this is done, indeed, 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 4 below.

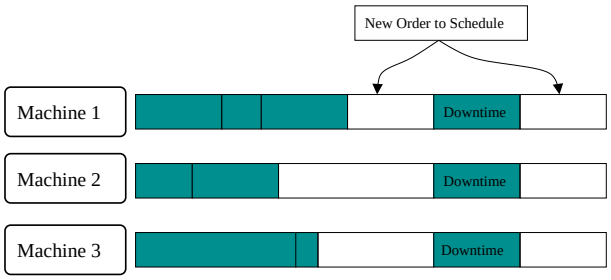

Fig. 4. Illustration of the Decoding Process

C. 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 ones 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. If $R(o)$ is a function that give an ordered list of machines that are capable of manufacturing the product associated with order o, then algorithmically, this operator can be represented as:

$i \sim U(0,k)$	`random machine index`
$j \sim U(0,\alpha(i))$	`random job on machine`
$a_i := a_i - o_j$	`remove job`
$M = \{m : R(m,o_j)\}$	`usable machines`
$i' \sim U(0,\|M\|)$	`new machine index`
$a_{i'} = a_{i'} \cup o_j$	`assign job`

- *Grouping*: This operator groups orders based on some common characteristic, such as wine colour, bottle type, or destination export country. If $C(o)$ is a function that gives the desired characteristic of an order, then algorithmically, this operator can be represented as:

`random machine index`
$i \sim U(0,k)$
`random job on machine`
$j \sim U(0,\alpha(i))$

`find left index of group`
$j_l = min\{h : 0 \leq h < j \text{ and } C(o_j) = C(o_h)\}$
`find right index of group`
$j_r = max\{h : j > h \geq \alpha(i) \text{ and } C(o_j) = C(o_h)\}$
`find index of similar group`
$j' \sim U(o,\alpha(i)) : j' \notin [j_l, j_r] \text{ and } C(o'_j) = C(o_j)$
`find left index`
$j'_l = min\{h : 0 \leq h < j \text{ and } C(o_j) = C(o_h)\}$

`remove first group`
$g_1 := a_i(j_l, j_r)$
$a_i := a_i - a_i(j_l, j_r)$

`insert next to second group`
$j_{insert} := j' - (j_r - j_l + 1)$
$a_i := a_i(0, j_{insert} - 1) \cup g_1 \cup a_i(j_{insert}, |a_i|)$

- *Order Prioritisation*: Orders that may be showing up as being produced late after decoding and individual are stochastically prioritised by moving them left in the decoding queue. If $D(o)$ is a function that gives the due date of an order o, and $A_c(o)$ is a function that gives its assigned completion time in the phenotype, then algorithmically, this operator can be represented as:

`random machine index`
$i \sim U(0,k)$
`random job on machine`
$j \sim U(0,\alpha(i))$

`check if completed late`
if $O(o_j) \geq A_c(o)$ then return

`move left to prioritise`
$j' \sim U(0, j-1)$
`adjust job list on machine`
$a_i := a_i - o_j$
$a_{i_{j'}} := o_j$

- *Job Mutation*: This is a pure mutation operator that moves a randomly selected job around without any preconditions, by swapping it with another job's position on the same machine. Algorithmically, this operator can be represented as:

`random machine index`
$i \sim U(0,k)$
`random job #1 on machine`
$j_1 \sim U(0,\alpha(i))$
`random job #2 on machine`
$j_2 \sim U(0,\alpha(i))$
`swap job #1 and job #2`
$o_{temp} := a_{i_{j_1}}$
$a_{i_{j_1}} := a_{i_{j_2}}$
$a_{i_{j_2}} := a_{temp}$

D. Solving the Dynamic Issues

- *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. The fitness function will 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 5.
- *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 cutoff are internally marked as manual assignments, and therefore behave in exactly the same way as described above for user-defined manual assignments.
- *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 evolutionary algorithm. This was accomplished using a process call *solution re-alignment*. Essentially, on each machine, all assigned orders, sorted in order by start date, are examined for any changes to the underlying orders. This then results in the assignment being shortened or lengthened as needed, all subsequent assignment 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 6

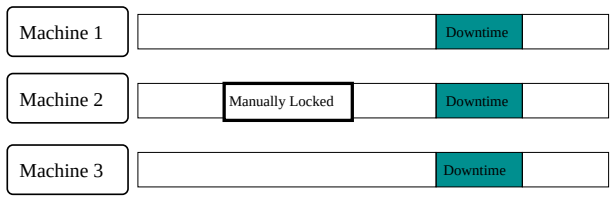

Fig. 5. Modified Initial Time Blocks for Manual Locks

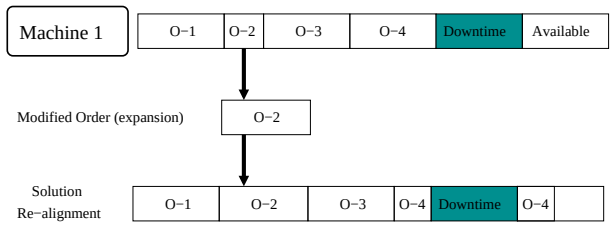

Fig. 6. Realigning a Solution

VII. Conclusions

In this paper, we looked at issues that arise when dealing with dynamic optimization problems in real-world applications. In particular, attention was focused on time-varying constraints that must be dealt with adequately in order to ensure that the solution produced is usable in an actual business environment.

The issues raised were exemplified by a case study based on an evolutionary algorithm 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 paper have passed the litmus test of suitability for real-world applications.

One of the key issues that was 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 modify the search space explored by the evolutionary algorithm. 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 schedule 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 will not be disrupted, and planning can proceed in a smooth and normal manner.

Overall, what this paper has illustrated is that in implementing a scheduling evolutionary algorithm for use in a practical commercial application, it is necessary to design 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 practical time-varying constraints to be easily considered by the core algorithm that finds an optimal schedule.

Acknowledgment

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] L. Davis, “Job shop scheduling with genetic algorithms,” in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 136–140.

[2] P. J. M. Van Laarhoven, “Job shop scheduling by simulated annealing,” *Operations research*, vol. 40, p. 113, 1992.

[3] L. Wang and D.-Z. Zheng, “A modified genetic algorithm for job-shop scheduling,” *International Journal of Advanced Manufacturing Technology*, 2002.

[4] R. S. M. R. Garey, D. S. Johnson, “The complexity of flowshop and jobshop scheduling,” *Mathematics of Operations Research*, 1976.

[5] R. Cheng, M. Gen, and Y. Tsujimura, “A tutorial survey of job-shop scheduling problems using genetic algorithms—i: representation,” *Comput. Ind. Eng.*, vol. 30, no. 4, pp. 983–997, 1996.

[6] T. Yamada and C. R. Reeves, “Solving the c_{sum} permutation flowshop scheduling problem by genetic local search,” 1998.

[7] E. Nowicki and C. Smutnicki, “A fast tabu search algorithm for the permutation flow-shop problem,” *European Journal of Operational Research*, vol. 91, no. 1, pp. 160–175, May 1996. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v91y1996i1p160-175.html

[8] E. Marchiori and A. Steenbeek, “An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling,” in *Scheduling, in Real World Applications of Evolutionary Computing. Springer-Verlag, Lecture Notes in Computer Science*. Springer, 2000, pp. 367–381.

[9] M. R. M. Ponnambalam, S.G., “A ga-sa multiobjective hybrid search algorithm for integrating lot sizing and sequencing in flow-line scheduling,” *International Journal of Advanced Manufacturing Technology*, 2003.

[10] T. Ray and R. A. Sarker, “Optimum oil production planning using an evolutionary approach,” in *Evolutionary Scheduling*, 2007, pp. 273–292.

[11] E. K. Burke and A. J. Smith, “A memetic algorithm to schedule planned maintenance for the national grid,” *J. Exp. Algorithmics*, vol. 4, p. 1, 1999.
[12] W. B. Langdon, *Scheduling planned maintenance of the national grid*. Springer, 1995, vol. 993, pp. 132–153.
[13] F. Martinelli, “Stochastic comparison algorithm for discrete optimization with estimation of time-varying objective functions,” *J. Optim. Theory Appl.*, vol. 103, no. 1, pp. 137–159, 1999.
[14] R. Tinós and S. Yang, “Genetic algorithms with self-organizing behaviour in dynamic environments,” 2007, pp. 105–127. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-49774-5_5
[15] S. Yang, “Explicit memory schemes for evolutionary algorithms in dynamic environments,” in *Evolutionary Computation in Dynamic and Uncertain Environments*, 2007, pp. 3–28.
[16] L. Schönemann, “Evolution strategies in dynamic environments,” in *Evolutionary Computation in Dynamic and Uncertain Environments*, 2007, pp. 51–77.
[17] A. Carlisle and G. Dozier, “Adapting particle swarm optimization to dynamic environments,” in *International Conference on Artificial Intelligence*, Las Vegas, NV, USA, 2000, pp. 429–434.