Kanban Scheduling System

Christian Colombo and John Abela

Department of Artificial Intelligence, University of Malta

Abstract. Nowadays manufacturing plants have adopted a demand-driven production control system also known as just-in-time production (JIT). This greatly reduces inventory costs because the buffers between the processes act as a blocking mechanism to indicate when production should stop and when eventually it can continue. One implementation of a JIT system is the Kanban system. Manufacturing in a JIT fashion, we face the challenge of flexibility to respond to changes in the customers' demands, while at the same time remaining cost-effective. This work attempts to introduce automation to the process of Kanban scheduling in a manufacturing plant environment with a multi-level part type. The proposed solution is a Memetic Algorithm (MA) which tries to optimize schedules such that they meet the deadlines of customer orders without causing buffer overflows, while at the same time keeping setup time to a minimum and placing free periods sensibly if these occur. The evaluation shows that this was achieved and the MA gave better results than the manual system currently in use.

1 Introduction

1.1 Overview — The Kanban System

The Kanban production system is a "pull" system where production takes place if, and only if, there is demand. The advantage of the system is that the stores (buffers) are virtually eliminated and the production becomes more responsive to the customers' demands.

Figure 1 shows how a customer's demand propagates through the production process. When an order is received from the customer, Process 3 starts to produce the finished goods (using the items from the upstream buffer). When a container of the items used is emptied from the buffer, it is returned to the Process 2 to refill it. Similarly, Process 2 starts to produce, using the items from the subsequent buffer. Again, the empty containers are returned to the previous process. Each container returned from a process to the previous has a "Kanban" associated to it which literally means a "card signal" in Japanese. The Kanban system was in fact introduced by Toyota in the 1950s. The term was coined for the fact that the Kanban actually "signals" a process to start production. [Jap89,Mon97a,JIN,Ols,MES01]

However, if each process in the production line starts to produce as soon as it receives an empty container, most of the time will be lost in setting up machines to produce the various items for each empty container. In a real-life situation,

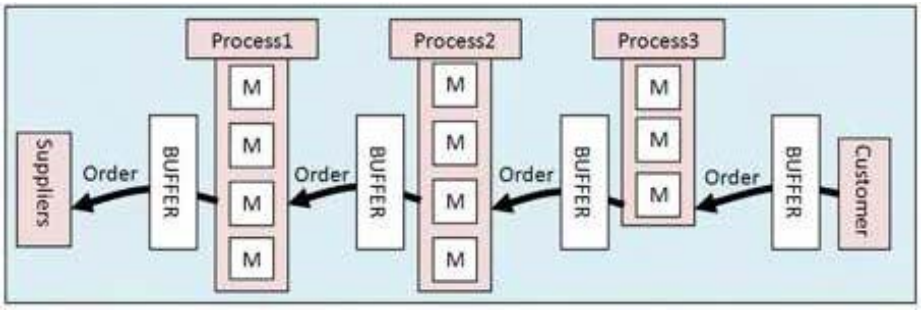

Fig. 1: "Pull" Production System

this problem is tackled delaying the start of production until a number of kanbans of the same item are received. Then, the setup-time needed becomes small (or acceptable) in relation to the run-time of the production. This also means that we should have buffers which are large enough to at least store the number of items produced from one runtime of the machine. Therefore the problem of scheduling kanbans is an optimization problem with aim of finding the best balance among the various costs which production entails [Mon97b].

1.2 Aims and Objectives

Main Aims The aim of this project is to develop an algorithm for automatically generating a schedule for the kanbans in a manufacturing plant. Furthermore, the project will help us study which factors in the manufacturing process attribute to higher production costs. This will help us generate an optimized schedule, trying to reduce the cost per item produced. The main aim of the project is to create an automated Kanban scheduling system which performs better than the manual system which is currently used.This will entail the creation of a model as close to reality as possible. Subsequently, an alogrithm will be developed which, given a data set will produce an optimized schedule. The schedule should be optimized with respect to various aspects. These include the setup time, the buffer balances and the free periods in the schedule.

Secondary Aims Without an adequate interface, it will be difficult to show the inputs, processing and results to the user. Therefore, a secondary aim is to create a verbose interface which allows the user to enter the necessary data, show the algorithm running and finally output the optimized schedules created. Another secondary aim is to test the algorithm with different parameters to find out the best settings to be used in practice. Furthermore, testing should be carried out to show that the automatic system performs better than the manual system currently in use.

2 Literature Review

2.1 NP Hard Problems

It is well known that scheduling problems are very challenging computational problems. However, real-life applications include a lot of different scheduling problems. These include school timetabling system, workers' roster and flight scheduling. Basically the scheduling problem is a search problem. In other words solving the scheduling problem can be described as searching through the space of all possible schedules and selecting the optimal schedule. The problem is that the number of possible schedules is usually exponential in relation to the size of the input. The problem is further complicated because the search space is not ordered and we may find one of the many local-minima, but we may still be far from the global-minimum [GJ79,Mit98,Mos89].

2.2 Heuristics to Solve NP Hard Problems

When tackling such NP Hard problems, rather than trying to obtain the optimum solution (which may take millions of years to find), we try to find a "relatively good" solution which will suffice for our real-life applications. There are many of these techniques which have been tried out successfully. Memetic Algorithms are one such example of successful techniques which have been implemented to solve (even though not optimally) a wide variety of scheduling problems [GJ79,Mos89].

2.3 Genetic Algorithms

Genetic Algorithms are algorithms inspired by natural evolution. The idea is that we try to emulate the process of natural selection to come to a better solution to a given problem. Therefore, we start with a population of individuals, all having different fitness (The fitness is a measure of how much the individual possesses the desired characteristics). Then we generate "new" individuals based on the existing ones by using genetic operators such as mutations and crossovers. Once these are generated, we choose which individuals will be kept for the next generation and the whole process is repeated again and again until there is sufficient increase in the fitness of the population [GJ79,Mos89].

2.4 Memetic Algorithms

Memetic Algorithms are an extension of Genetic Algorithms, where we keep the same idea, but we try to improve the search of solutions by limiting the random element of the search and trying to find local maxima by using appropriate optimization techniques. In other words, we try to improve the individuals of the population, not simply generate the individual using the genetic operators. This method will generally make the search better, because now the individuals will all represent a local minimum in the search space, not just any possible point of the space [GJ79,Mos89].

3 Design

3.1 Generating Random Schedules

The initial step of a basic evolutionary algorithm is to generate a population of new random individuals (in our case schedules) [Mit98]. Generating random schedules is not an easy task since we are trying to emulate the real-life system where a number of autonomous "entities" are acting at the same time. These "entities" are actually the different sections which are triggering each other to work: starting from the last section till the starting section. It is important to note that we should not issue all the Kanbans from the beginning of the scheduling process, but rather let the Kanbans be issued automatically once there is space in the buffer.
Using this system we are automatically "going down" the bill-of-material tree. This is exactly what happens in a real-life situation [Mon97b] and is further explained in Figure 2.

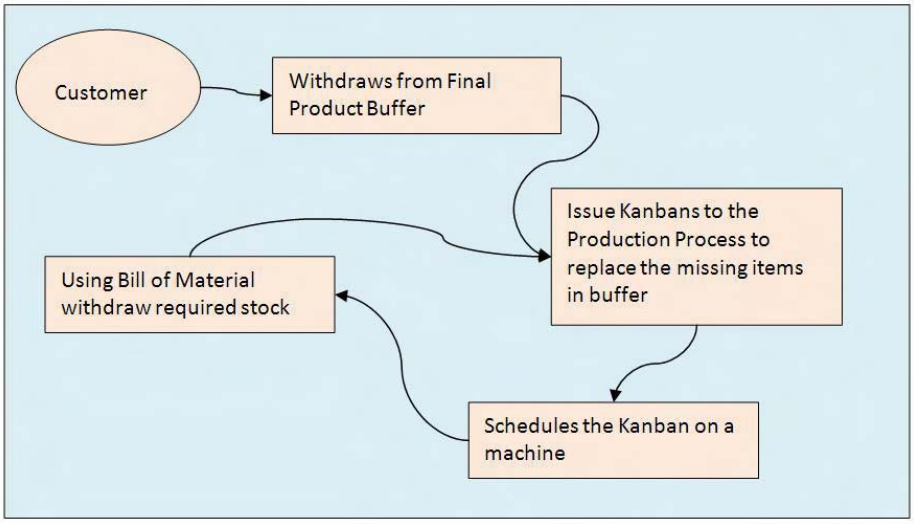

Fig. 2: Withdrawal cycle

3.2 The Building Blocks of a Memetic Algorithm

When designing a Memetic Algorithm (or a genetic algorithm), the genetic operations should be designed with great care, since these are used in each generation [Mit98,Mos89] and hence are crucial in the evolution of the population. Another design aspect of any genetic algorithm is the choice of a fitness function [Mit98]. It must be ensured that this function includes all the aspects that we need to

optimize. If an aspect is not included in the function, the individuals cannot not be optimized with respect to that aspect.
The following sections will describe the design of the fitness function and the operations used for the Memetic Algorithm.

3.3 The Fitness Function

The aspects considered in the fitness function are the following:

1. The total setup time used by the machines.
2. A penalty is also given to each time unit during which a buffer is an invalid state (either over the maximum or below the minimum). However, in this case the penalty also depends on the amount by which the buffer is invalid.
3. Furthermore, an extra penalty is assigned for the amount by which buffers are still in an invalid state at the end of the schedule generated.
4. Another part of the penalty is assigned for the number of times a machine starts and stops. The reason for this penalty is that in practice it is undesirable that a machine starts and stops with free periods in between.
5. Another penalty is assigned when the machines of a section do not start or stop together. From a practical point of view [Mon97b], it is undesirable that there is only one machine working in a section when there are three machines, which can therefore reduce the production lead time.

3.4 Mutations

The designed mutation can use one or more genetic operations for any number of times. The operations designed to be used be the mutation may be one of the following:

1. The first operation is to swap two jobs in one section. For example: a job on machine m1 at time t1 is swapped with machine m2 at time t2. The purpose of this operation is basically to try to reduce the amount of setup time for the machines.
2. The second operation also swaps a job, only that this time not with another job but with a free period. This operation will help in removing the unwanted free periods which are in the middle of jobs in a schedule.
3. Another operation is the complete removal of a job from a schedule. Sometimes it is better to reduce the production to eliminate buffer overflows and this operation in fact simply removes a job to reduce production.
4. The fourth operation is to insert a free period before a job. The purpose of such an operation is to increase the probability of having schedules which start the jobs of a section simultaneously at the same time.
5. Similar to the previous, this operation simply removes a free period from the schedule. The purpose is to reduce the number of times the machines are stopped and restarted, and may also help to start the machines in the section at the same time (as in the previous point).

3.5 Local Search

Local search is an optimization done on an individual of the population. There are two strategies which aim to improve different aspects of the schedules.

Strategy 1 — Reducing Buffer Penalties When we analyze a buffer to calculate the penalties of overflow or underflow, we can deduce which production jobs should be moved earlier or later to eliminate the penalties. The idea is that if we have encountered an underflow earlier, then the jobs creating the overflow should have occurred earlier; they have been "delayed". Inversely, if we meet an overflow which was not preceded by an underflow, then the jobs should be "moved" to a later time; they are "early".

Strategy 2 — Reducing "Imbalanced" Sections Penalties The second strategy has the aim of reducing the number of imbalanced schedules of sections. This is done by examining the jobs which are being produced "alone" i.e. no other machine in the same section is working while that job is being carried out. Such jobs should be re-scheduled somewhere else where there are more machines at work to avoid sections working inefficiently.

3.6 Crossovers

Two types of crossovers are proposed: one is purely random while the other tries to take the best out of the two schedules randomly selected from the population.

Random Crossovers The random crossover generation works by selecting two random schedules from the population (after selection has taken place) and then for each machine, randomly decides whether to use the first or the second schedule. Once this is completed, we need to re-generate the buffer balances since neither the buffers of the first schedule nor those of second schedule, can be used as the buffers of the newly generated schedule. The purpose of using this purely random approach is that sometimes it is difficult to know which parts of which schedule will yield an overall better fitness. Hence, this random approach may sometimes give good results depending on chance.

"Intelligent" Crossovers The other type of crossover attempts to take the best out of the two randomly selected schedules. This is done by treating a schedule on a "section-by-section" basis, and calculate each section's penalty. Then, we simply select the sections from the randomly selected schedules with the least penalty to form a new schedule with some sections possibly from either parent.

3.7 Method of Selecting Individuals

One of the main genetic operators is "selection" i.e. the way the individuals are selected to be allowed to remain in the population and generate offsprings [Mit98]. From the selection strategies listed in the literature ([JIN] p.166) we will choose a combination of the "Tournament Selection" method and the "Elitism" method. The purpose of using the "Elitism" is that it allows us to "secure" some of the best individuals of the population while that of using the Tournament selection is that is allows some of the non-best individuals to pass to the next generation so that we keep more "variety" in the population.

3.8 The Memetic Algorithm as a Whole

The Memetic Algorithm starts by first creating an initial population of random schedules. Then, a generation starts; each generation consists of the following summarized steps:

1. Apply genetic operators of mutation and crossover to produce new individuals.
2. Apply local search to improve individuals.
3. Use the selection algorithm to select which individuals will be kept to the next generation.

4 Implementation

The architecture selected for the implementation of the algorithm is the .NET framework using the C# language. The object-oriented design together with a user-friendly graphical user interface where implemented. Furthermore, the designed algorithms (described in the previous section) were implemented successfully. The following is an overview of some important implementation details.

4.1 Schedule Structure

There are two main parts of a schedule: (1) the machines' schedule and (2) the buffers' schedule. The machines' schedule will contain the sequence of jobs which each machine will process throughout a period of time. On the other hand, the buffers' schedule contains the activities of the buffers. This is obviously related to the machines' schedule since the material is needed for production (and hence withdrawn from the buffers) and products are placed in the buffer as a result of production (hence inputting the products in the buffers).
The machines' schedule is stored as a list of periods in an array. Therefore in order to schedule a new job, we must find a free period, update the affected transitions and insert free periods if the original free period is longer than the required time. This is shown in Figure 3.

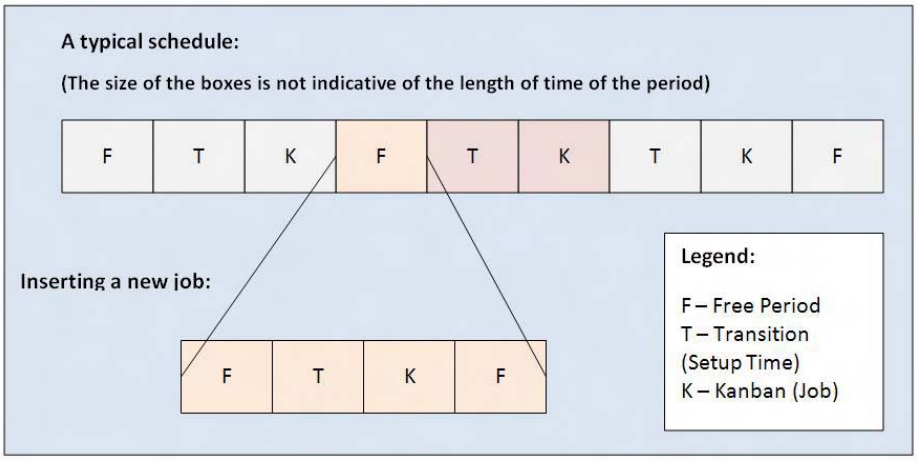

Fig. 3: Inserting a new job in the schedule

5 Testing and Results

Testing was performed on data as close to reality as possible. An example of the results obtained is shown in Figure 4.

One should note that the schedule contains minimal setup time for machines (denoted by the pink (dark grey) boxes) and all the rest of the time units are allotted to jobs (white boxes). The green (light grey) boxes denote a change of shift. This means that the machine time is being efficiently utilized. At the same time the scheduling system also generates the schedule for the buffers. This shows what products are entering and leaving the buffer at any time. A sample is shown in Figure 5.

In this case one should note that all the boxes are white or light purple (very light grey) (except for the green boxes (light grey) which denote a change of shift). This means that there are no buffer overflows and even more importantly no buffer underflows. In other words, all the deadlines have been met without wasting material by leaving it idle in the buffers with the possibility of not finding demand for it.

When the result obtained by the Kanban scheduling system was compared to a typical result obtained by the manual system it was found out that the automated system performs better than the manual system. The main advantage of using the automated system was that all the orders were fit in the schedule, while the manual system was unable to fit in all the orders in time. However, it was noted that the manual system may do better in terms of setup time used. Nevertheless, considering the overall penalty, the automated system performed significantly better. Furthermore, one must consider that the manual system requires a lot of human effort while in the case of the automated system the effort is done by the computer.

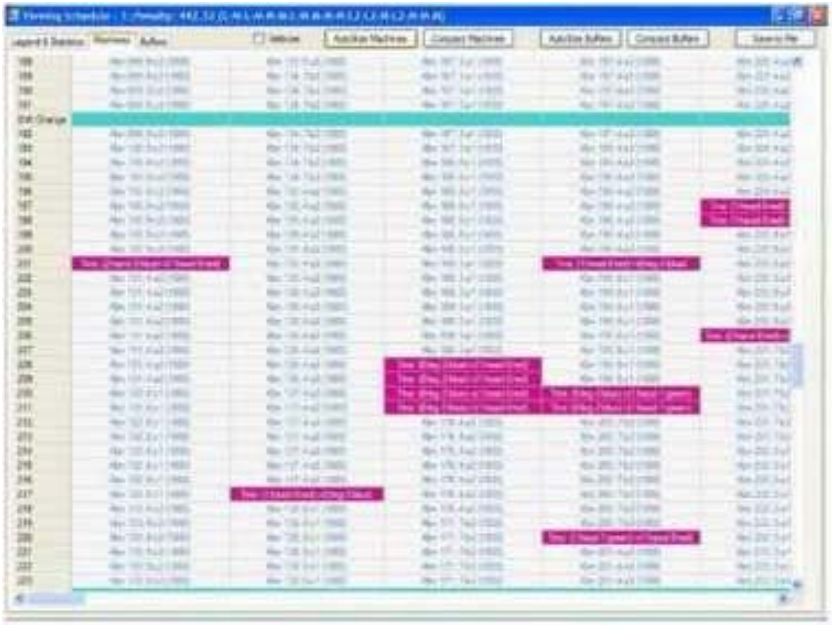

Fig. 4: A sample of a machine's schedule obtained

Fig. 5: A sample of the buffers' schedule

6 Conclusion

We conclude that Memetic Algorithms are very much suited for real life optimization problems such as the Kanban scheduling problem. In the current project the problem was limited to certain boundaries such as no machine maintenance, which in practical terms cannot be overlooked. However, the proposed solution and the tests carried out show that there is a great potential in using evolutionary algorithms in helping schedulers to produce more cost effective schedules in a manufacturing environment.

References

[GJ79] Michael R. Garey and David S. Johnson. Computers and intractability. Technical report, Bell Telephone Laboratories, Inc., 1979.

[Jap89] Japan Management Association. Kanban just-in-time at toyota: management begins at the workplace. Technical report, Productivity Press, 1989.

[MES01] Richard P. Marek, Debra A. Elkins, and Donald R. Smith. Understanding the fundamentals of kanban and conwip pull systems using simulation. In *Proceedings of the 2001 Winter Simulation Conference*, 2001.

[Mit98] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[Mon97a] Yasuhiro Monden. Toyota production system: An integrated approach to just-in-time. Technical Report 3rd Edition, Institute of Industrial Engineers, 1997.

[Mon97b] Yasuhiro Monden. Toyota production system: An integrated approach to just-in-time. Technical report, Institute of Industrial Engineers, 1997.

[Mos89] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts towards memetic algorithms. Technical report, California Institute of Technology, 1989.

[Ols] Johan Olsson. Kanban — an integrated jit system. Management Systems Consulting, Inc. `http://www.msc-inc.net/Documents/Kanban\_Integrated\_JIT\_System.htm` Last accessed 7th March 2007.

[JIN] Just-in-time / kanban. `http://www.epa.gov/lean/thinking/Kanban.htm` Last accessed 7th March 2007.