Kimms, Alf

A cellular automaton based heuristic for multi-level lot sizing and scheduling

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 331

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Kimms, Alf (1993) : A cellular automaton based heuristic for multi-level lot sizing and scheduling, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 331, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/155409

Nr. 331

A Cellular Automaton Based Heuristic for Multi-Level Lot Sizing and Scheduling

A. Kimms

November 1993

Dipl.-Inform. Alf Kimms, Lehrstuhl für Produktion und Logistik, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstraße 40, 24118 Kiel, Germany

Abstract: Cellular automata were used to model and to simulate phenomena in the area of physics, biology and medicine. In this paper it is now shown how the idea of cellular automata can be applied to optimization problems as well. As an example a cellular automaton is used as a basis for solving multi-level lot sizing and scheduling problems to suboptimality. We will furthermore give an outline of a proof that any genetic algorithm can be interpreted as a cellular automaton.

Keywords: Production planning, lot sizing, scheduling, PLSP, multi-level, cellular automata, hybrid heuristics, genetic algorithms.

1 Introduction

Integer and combinatorial optimization belong to the most challenging subjects in operations research. While a mixed-integer program can often be formulated in order to describe a certain problem mathematically, the optimal solution of a complex real-world problem can more often not be computed by means of standard solvers. To attack this problem a lot of special purpose heuristics were developed which render it possible to solve a narrow class of problems to suboptimality (or more optimistically spoken to find an ε-approximate solution [Nemhauser and Wolsey 1988]). The advantages of such heuristics when compared to exact methods are a short response time and more important the capability to find suboptimal but feasible solutions where exact methods fail. These heuristics are usually classified as construction, improvement, mathematical programming, partitioning, restriction or relaxation methods [Sinclair 1993, Zanakis et al. 1989]. More recent so-called meta algorithms [Hertz and de Werra 1990, Sinclair 1993] can be combined with existing approaches to guide the solution process. Some of these methods are known as simulated annealing [Collins et al. 1988, Johnson et al. 1989 and 1991], genetic algorithms [Dorndorf and Pesch 1992, Goldberg 1989, Holland 1975, Liepins and Hilliard 1989, Mühlenbein et al. 1988] and tabu search [Faigle and Kern 1992, Glover 1989, 1990a and 1990b, Hertz and de Werra 1990], respectively. In the sequel we will introduce the idea of cellular automata [Toffoli and Margolus 1988, Wolfram 1986] and as an example we will show how this idea can be used as a basis for solving a hard production planning problem to suboptimality. The next section will thus be dedicated to the fundamentals of cellular automata. To relate this approach to other work, we will subsequently point out that any genetic algorithm can be interpreted as a cellular automaton. Afterwards, we will present the multi-level, single-machine lot sizing and scheduling problem which will be the subject of our consideration. We will then develop a cellular automaton based heuristic and give some computational results.

2 Cellular Automata

A deterministic cellular automaton is defined by six components:

- A (finite or infinite) lattice $\Lambda \subseteq \mathbf{Z}^m$ of so-called cells. The lattice can be of any dimension $m \in \mathbf{N}^+$. Each cell is uniquely identified by a tuple of the form $(\lambda_1, \ldots, \lambda_m) \in \Lambda$.

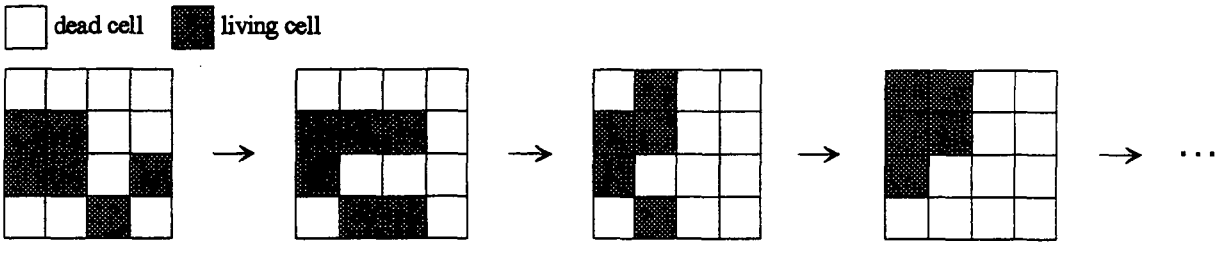
- A state-space Σ. Each cell $(\lambda_1, \ldots, \lambda_m) \in \Lambda$ is in a well-defined state $\sigma [(\lambda_1, \ldots, \lambda_m)] \in \Sigma$. The state of the whole lattice is denoted by $\sigma [\Lambda] \in \Sigma^{|\Lambda|}$.
- An initial state of the lattice $\sigma_0 [\Lambda] \in \Sigma^{|\Lambda|}$, i.e. an initial state for each of the cells.
- A family of state-transition functions $\theta_{\sigma[\Lambda]} : (\Lambda \times \Sigma) \rightarrow \Sigma$.

  (A stochastic state-transition rule would lead to the notion of stochastic cellular automata.)
- A neighborhood function $\nu : \Lambda \rightarrow \wp (\Lambda)$ where $\wp$ denotes the powerset operator.
- A global clock.

The automaton then works as follows: Starting with the initial state of the lattice, the state-transition function $\theta_{\sigma[\Lambda]}$ that corresponds to the current state $\sigma [\Lambda]$ is simultaneously applied to all cells at every clock tick where the (new) state of a cell $(\lambda_1, \ldots, \lambda_m) \in \Lambda$ depends on the (old) states of the cells in the neighborhood $\nu [(\lambda_1, \ldots, \lambda_m)]$.

Example: One of the most well-known cellular automata is the "Game of Life" invented by Conway in 1968 [Gardner 1970]. He used a two-dimensional lattice to study the life cycles of (biological) cells: Each cell can be either dead or alive. A potential scenario is that a living cell survives if it is surrounded by two or three other living cells otherwise it dies. A dead cell can be revived (i.e. a new cell is born) if it is surrounded by exactly three living cells. Figure 1 illustrates the behavior of this automaton.

Figure 1: "Game of Life"



3 Well-Known Stochastic Cellular Automata: Genetic Algorithms

Genetic algorithms [Goldberg 1989, Holland 1975] were deduced from biology where in a population of life forms the fittest individuals survive while the weakest die. The basic idea of these algorithms is to code a feasible solution of a problem as a string (which is termed individual or chromosome while a single character within such a string as called a gene). The task is to find an individual which defines a "good" solution to a given problem. To do so, a population of size $P \in \mathbb{N}^+$ (which is the set of "parent"-individuals) is considered. By using some basic genetic operators (which are to be defined), a number of $C \in \mathbb{N}^+$ new individuals (which is the set of "child"-individuals) can be constructed. A fitness function is then used to decide which individuals out of the pool of P+C individuals survive to form a new population (or generation) of size P. Noteworthy to say, that "parent"-individuals may survive for several generations if they are very fit. This process is repeated over and over again in search for an individual with utmost fitness. More formally, a genetic algorithm is defined as follows:

- $P \in \mathbb{N}^+$, the size of the "parent"-population.
- $C \in \mathbb{N}^+$, the size of the "child"-population.
- $L \in \mathbb{N}^+$, the number of genes which form an individual, i.e. the length of a string that represents a solution to a problem instance.
- V, the state-space of the genes. The state of an individual is thus a member of the L-fold cross product of V which is denoted in the usual way as V^L.

  (For the sake of simplicity we assume, that the states of all the genes which build a string are members of a common set V. In the case of different domains at different positions in the string of genes, one could define V as the union of all domains.)
- An initial population of size P.
- A fitness function *fitness* : $V^L \rightarrow \mathbb{R}$ which assigns a fitness value to each individual. On the basis of this value a new "parent"-population is selected.
- A set of basic genetic operators. Usually, the following operators are used:

  crossover : $(V^L \times V^L) \rightarrow \wp (V^L)$ which "cuts" two individuals at the same position somewhere in the middle (this position is chosen at random) and appends the last part of the first individual to the first part of the second individual and vice versa to build two new individuals.

  mutation : $V^L \rightarrow V^L$ which changes the state of exactly one gene (chosen at random) to produce a new individual.

The individuals to which the basic genetic operators are applied to are chosen at random out of the set of "parent"-individuals.

We will now point out that genetic algorithms can be interpreted as stochastic cellular automata:

Let $\Lambda = \{ 1, \ldots, P, \ldots, P+C \} \times \{ 1, \ldots, L \}$ and $\Sigma = V$. This is to say, that a genetic algorithm is a two-dimensional cellular automaton where each "row" of this automaton represents an individual and each cell represents a gene. The first P "rows" will be used to define the "parent"-population. The initial state of the cellular automaton must thus be defined by the initial population of the genetic algorithm for the first P "rows" and can arbitrarily be defined for the last C "rows". The order in which the initial population is mapped onto the first P "rows" is irrelevant. The neighborhood of a cell $(\lambda_1 , \lambda_2) \in \Lambda$ is defined as $\nu [(\lambda_1 , \lambda_2)] = \Lambda$. To define a family of stochastic state-transition rules $\theta_{\sigma [\Lambda]}$ we first introduce a convenient notation: For $state \in \Sigma^{|\Lambda|}$ and $\lambda \in \{ 1, \ldots, P+C \}$, $state [\lambda]$ is called a projection and denotes the λ-th "row" of the lattice Λ where $state [\lambda] \in \Sigma^L$. The expression $state [(\lambda_1,\lambda_2)] (= \sigma [(\lambda_1,\lambda_2)] \in \Sigma)$ is called a projection as well and denotes the λ_2-th entry in the λ_1-th "row" of the lattice Λ. For each cell $(\lambda_1 , \lambda_2) \in \Lambda$ we can now define $\theta_{\sigma [\Lambda]}$ as follows:

$$\theta_{\sigma [\Lambda]} [(\lambda_1 , \lambda_2) , \sigma [(\lambda_1 , \lambda_2)]] = select (trans (\sigma [\Lambda])) [(\lambda_1,\lambda_2)]$$

where

$$select : \Sigma^{|\Lambda|} \rightarrow \Sigma^{|\Lambda|}$$

is some kind of a sort function that reorders all the "rows" of a lattice Λ such that the P individuals that form the (new) "parent"-population of the next generation can be found within the first P "rows". The selection of the new "parent"-population can thereby be done either deterministically, so that

$$fitness (select (\sigma [\Lambda]) [\lambda]) \leq fitness (select (\sigma [\Lambda]) [\lambda']) \text{ if and only if } \lambda \geq \lambda'$$

holds for all $\lambda, \lambda' \in \{ 1, \ldots, P+C \}$, or stochastically, so that the higher the fitness value of an individual the more probable does this individual appear as a "parent" of the next generation.

Furthermore,

$$trans : \Sigma^{|\Lambda|} \rightarrow \Sigma^{|\Lambda|}$$

is a procedure that computes C new "child"-individuals. This is done by choosing one of the basic genetic operators at random. Suppose that the selected operator combines $n \in \mathbb{N}^+$ individuals to produce a new one. The *trans* procedure then selects n "rows" out of the first P "rows" of the lattice and applies the genetic operator to these individuals. Another genetic operator is then chosen at random and the whole process is repeated until a total of C new individuals is computed. These individuals are eventually used to update the last C "rows" of the lattice whereby the sequence in which the new individuals are used to fill up the lattice is of no relevance.

4 The Multi-Level Lot Sizing and Scheduling Problem

The focus of our attention is a non-trivial short-term production planning problem, i.e. the proportional lot sizing and scheduling problem (PLSP) [Drexl and Haase 1992, Haase 1993, Kimms 1993]: Several items are to be produced on one machine in a multi-level production process. We consider a finite planning horizon which is divided into several discrete time periods. Before one item can be produced the machine has to be setup for this particular item. Setting up the machine causes item specific setup costs. We assume that at most one setup may occur in a certain period of time and that the capacity of the machine is limited while the production of one item consumes an item specific quantity of capacity units. It may therefore be necessary to produce some items in periods much earlier than the demands for these items are to be met. In this case the items are to be stored in inventory which causes item specific holding costs for every period that the items are held in inventory. Demands are assumed to be deterministic but dynamic and shortages are not allowed. The objective is to find a cheap and feasible production plan.

More precisely, the problem is defined by the following mixed-integer program [Kimms 1993]:

$$\min \sum_{t=1}^{T} \sum_{j=1}^{J} (s_j x_{jt} + h_j I_{jt}) \qquad (1)$$

subject to

$$I_{jt} = I_{j(t-1)} + q_{jt} - d_{jt} - \sum_{i \in S(j)} (a_{ji} q_{it}) \qquad (j = 1 .. J, t = 1 .. T) \qquad (2)$$

$$I_{jt} \geq \sum_{\tau=t+1}^{\min\{t+v_j, T\}} \sum_{i \in S(j)} (a_{ji} q_{i\tau}) \qquad (j = 1 .. J, t = 0 .. T-1) \qquad (3)$$

$$\sum_{j=1}^{J} y_{jt} \leq 1 \qquad (t = 1 .. T) \qquad (4)$$

$$B (y_{jt} + y_{j(t-1)}) - q_{jt} \geq 0 \qquad (j = 1 .. J, t = 1 .. T) \qquad (5)$$

$$x_{jt} - y_{jt} + y_{j(t-1)} \geq 0 \qquad (j = 1 .. J, t = 1 .. T) \qquad (6)$$

$$\sum_{j=1}^{J} (p_j q_{jt}) \leq C_t \qquad (t = 1 .. T) \qquad (7)$$

$$y_{jt} \in \{ 0 , 1 \} \qquad (j = 1 .. J, t = 1 .. T) \qquad (8)$$

$$I_{jt} \geq 0, q_{jt} \geq 0, x_{jt} \geq 0 \qquad (j = 1 .. J, t = 1 .. T) \qquad (9)$$

where

a_{ji} is the "gozinto-factor", i.e. the quantity of item j that is needed to produce one item i;

B is a large number greater than $\frac{\max \{ C_t \mid t = 1 .. T \}}{\min \{ p_j \mid j = 1 .. J \}}$,

C_t is the capacity of the machine in period t;

d_{jt} is the (external) demand for item j in period t;

h_j are the costs for holding one item j one period in inventory;

I_{jt} is the quantity of item j held in inventory at the end of period t (I_{j0} is the initial inventory);

J is the number of items;

p_j is the amount of capacity consumed by producing one item j;

q_{jt} is the quantity of item j to be produced in period t;

s_j are the setup costs for item j;

S(j) is the set of successors of item j, i.e. the set of items i where $a_{ji} > 0$;

T is the number of periods;

v_j is the (integral) lead time of item j ($v_j \geq 1$);

x_{jt} is a (binary) variable indicating whether a setup for item j occurs in period t ($x_{jt} = 1$) or not ($x_{jt} = 0$);

y_{jt} is a binary variable indicating whether the machine is setup for item j at the end of period t ($y_{jt} = 1$) or not ($y_{jt} = 0$) (y_{j0} is the initial setup state).

While (1) defines the objective to minimize the sum of setup and holding costs, (2) describe the inventory balances. The constraints (3) ensure that internal demands are met promptly with respect to positive lead times. The restrictions (4) (in combination with (8)) make sure that the setup state of the machine is uniquely defined at the end of each period. The fact that the machine must be setup for an item before this particular item can be manufactured is expressed by (5) where those periods of time in which setups take place are spotted by (6). The machine capacity must not be exceeded which is formulated by (7). (8) and (9) define decision variables to be non-negative.

5 Specification of a Heuristic

Before we will give a heuristic procedure that can be used to look for a feasible (and cheap) production plan let us first work out the point of view that makes cellular automata attractable for solving PLSP-instances. To do so we should have a closer look at what the idea of lot sizing and scheduling is about: Lot sizing means to bundle up demands for the same item in order to save setup costs. By doing so in a capacitated environment it usually happens that the production of such a lot lasts several succeeding time periods. The act of sequence planning to find a production plan that fits into a given time frame and that respects the multi-level product structure is termed scheduling while at most one item is scheduled at each time spot, i.e. if a unique setup state is defined. The combination of both lot sizing and scheduling tends to find a production plan where the setup state does not flicker, i.e. the unique setup state is kept the same over a couple of succeeding periods before it changes.

The underlying idea of the cellular automaton based approach is to compute a setup state pattern (by means of a cellular automaton) first and to generate a production plan that fits to the setup state pattern afterwards. Repeating this process leads to a number of different production plans from which the best one is chosen. In the sequel we will describe this two-step scheme in more detail starting with the generation of setup state patterns.

To compute several setup state patterns for problem instances with J items and T periods we use a stochastic cellular automaton defined as follows: $\Lambda = [\,1, \ldots, J\,] \times [\,1, \ldots, T\,]$ is the finite lattice of cells. Each cell can be either dead or alive, i.e. the state-space is $\Sigma = \{$ dead, alive $\}$. A living cell $(j, t) \in \Lambda$ is interpreted as item j may be produced in period t, whereas a dead cell $(j, t) \in \Lambda$ is interpreted as item j must not be produced in period t. This is to say, that the state of the cells defines for each item a mask which enables (but does not enforce) the production in some periods and forbids the production in some other periods.

As pointed out before, two relevant aspects are to be modelled: Keeping up the setup state over several periods of time and a unique definition of the setup state at the end of each period. An appropriate definition of the neighborhood function ν and the family of stochastic state-transition rules $\theta_{\sigma[\Lambda]}$ allows us to take these two aspects into account:

The first criterion, i.e. the fact that the setup state tends to be kept up over several time periods, can by paid attention to by watching the neighborhood

$$\nu_1[(j,t)] = \{(j',t') \in \Lambda \mid (j=j') \wedge (\,|\,t-t'\,|\,=1)\}.$$

This is to say, that if we have to decide whether or not to allow item j to be produced in period t while computing a new setup state pattern, we look at what decision we have made for item j in periods t-1 and t+1 for the old setup state pattern. If item j was allowed to be produced neither in period t-1 nor in period t+1, a permission to produce item j in period t might lead to a better production plan. If the production of item j was allowed in only one of these periods, it might be a good idea to permit production in period t, too, because production may last longer than one period. If both setup states in period t-1 and t+1 were set, it might be a good choice to allow item j to be produced in period t as well, since a non-preemptive production may then take place.

Following this idea, we discriminate three cases. These three cases are:

(1) $\left| \{ (j', t') \in \nu_1 [(j, t)] \mid \sigma [(j', t')] = \text{alive} \} \right| = 0$,

(2) $\left| \{ (j', t') \in \nu_1 [(j, t)] \mid \sigma [(j', t')] = \text{alive} \} \right| = 1$, and

(3) $\left| \{ (j', t') \in \nu_1 [(j, t)] \mid \sigma [(j', t')] = \text{alive} \} \right| = 2$.

To each of these cases we assign a certain probability (π_{a1}, π_{a2} and π_{a3}, respectively) for a living cell to stay alive and a certain probability (π_{r1}, π_{r2} and π_{r3}, respectively) for a dead cell to be revived. The choice of these probabilities is quite unclear. We could either use a fixed probability assignment to compute new generations of setup state patterns or we could modify the probability assignment after each state-transition of the lattice where there are manyfold ways to modify the assignments (pure random, guided by local search techniques, ...). Given a particular test bed, computational studies are to be done to find out how to proceed.

The second aspect concerning the fact that the setup state is uniquely defined at the end of each period is taken into account by considering the neighborhood

$$\nu_2 [(j, t)] = \{ (j', t') \in \Lambda \mid (j \neq j') \wedge (t = t') \}.$$

The decision whether or not to permit production of item j in period t depends on how many other items own a permission to be produced in period t. If no other items were allowed to be produced, we can feel free to assign such a permission to item j. If exactly one other item was allowed to be produced, item j may be produced as well because the PLSP-model allows one changeover in period t. If two or more other items had a permission to be produced, it might be clever to disallow production of item j in period t. We thus distinguish three cases to each of which a certain probability is assigned for a living cell to stay alive (μ_{a1}, μ_{a2} and μ_{a3}, respectively) and a dead cell to be revived (μ_{r1}, μ_{r2} and μ_{r3}, respectively). The cases that we take into consideration are:

(1) $\left| \{ (j', t') \in \nu_2 [(j, t)] \mid \sigma [(j', t')] = \text{alive} \} \right| = 0$,

(2) $\left| \{ (j', t') \in \nu_2 [(j, t)] \mid \sigma [(j', t')] = \text{alive} \} \right| = 1$, and

(3) $\left| \{ (j', t') \in \nu_2 [(j, t)] \mid \sigma [(j', t')] = \text{alive} \} \right| \geq 2$.

Again, it is quite unclear how to choose the probabilities and once more computational studies should be used to find an answer when applied to a particular problem instance.

The neighborhood of a cell $(j, t) \in \Lambda$ can now be defined as

$$\nu [(j, t)] = \nu_1 [(j, t)] \cup \nu_2 [(j, t)].$$

As a stochastic state-transition rule $\theta_{\sigma [\Lambda]}$ we use the following scheme: For each cell $(j, t) \in \Lambda$ we check both neighborhoods $\nu_1 [(j, t)]$ and $\nu_2 [(j, t)]$ in separation to find out which of the cases defined above holds. Suppose that $\xi \in \{ 1, 2, 3 \}$ holds when we test $\nu_1 [(j, t)]$ and $\zeta \in \{ 1, 2, 3 \}$ holds when we test $\nu_2 [(j, t)]$. Given certain probabilities $\pi_{a\xi}$ and $\mu_{a\zeta}$ for a living cell to stay alive in case ξ and in case ζ, respectively, we assume $\pi_{a\xi}\, \mu_{a\zeta}$ to be the probability of a living cell to stay alive if both neighborhoods are considered in combination. Analogously, we assume $\pi_{r\xi}\, \mu_{r\zeta}$ to be the probability of a dead cell to be revived. To compute the new state of the cell, we then draw two random numbers ω_1 and ω_2 where $0 \leq \omega_1, \omega_2 \leq 1$ are uniformly distributed. The state of the cell is transformed by the following rules:

$$\theta_{\sigma[\Lambda]}[(j,t),\text{alive}] = \begin{cases} \text{alive} & \text{if } \omega_1\omega_2 \le \pi_{a\xi}\,\mu_{a\zeta} \\ \text{dead} & \text{otherwise} \end{cases}$$

and

$$\theta_{\sigma[\Lambda]}[(j,t),\text{dead}] = \begin{cases} \text{alive} & \text{if } \omega_1\omega_2 \le \pi_{r\xi}\,\mu_{r\zeta} \\ \text{dead} & \text{otherwise} \end{cases}$$

The initial state of the lattice is arbitrarily chosen, e.g. all cells are assumed to be alive when we start.

To complete the description of our PLSP-heuristic it has to be defined how to (try to) find a feasible production plan once that a setup state pattern as determined. The basic idea is to employ a backward oriented regret based heuristic [Drexl 1991]. While moving backwards from period T to period one items are scheduled in the following way: Given a demand matrix $(d_{jt})_{\substack{j=1..J \\ t=1..T}}$ the cumulated demand for item j in period t (denoted as CD_{jt}), i.e. the sum of demands that are not met yet, is defined as $CD_{jt} = \sum_{\tau=t}^{T} d_{j\tau} - \sum_{\tau=t+1}^{T} q_{j\tau}$. Within each period an item with a positive cumulated demand is scheduled where an item j may only be scheduled in period t if the state $\sigma[(j,t)]$ = alive. The production quantity in period t of the selected item j is computed by $q_{jt} = \min\{CD_{jt}, \lfloor \frac{C_t}{p_j} \rfloor\}$. The demand matrix is to be updated by the internal demand for directly predecessing items in order to handle multi-level structures correctly, i.e. $d_{i(t-v_{ij})} = d_{i(t-v_{ij})} + a_{ij}\, q_{jt}$ for all $i \in \{1,...,J\}$. If the available capacity of the machine does not suffice to produce the whole cumulated demand in period t, the production for item j is continued in period t-1 if and only if $\sigma[(j,t\text{-}1)]$ = alive. In the case that the whole cumulated demand can be produced in period t and that there are some capacity units left a second item is scheduled likewise. This procedure goes on until period one is reached. A feasible production plan is eventually found if no item j remains with a positive cumulated demand.

The details of this algorithm can be found in [Kimms 1993] and shall not be reviewed. However, we should briefly mention the way an item is selected for production. This choice is based on a so-called regret measure that estimates the regret not to schedule an item in a certain period. Our regret measure reflects two aspects: The costs that are incurred when an item is not scheduled and the risk of infeasibility. Four criteria do (in our opinion) influence the regret measure.

(a) If the cumulated demand of item j is not scheduled in period t we have to pay additional holding costs.

(b) If item j is produced at the beginning of period t+1 then we may save setup costs if a production of item j takes place at the end of period t as well.

(c) Preceding items must be manufactured before item j can be produced. If item j is scheduled in early periods it may therefore happen that the depth of the product structure exceeds the time frame which means that no feasible production plan can be found. The depth of item j (denoted as dep_j) can be computed as a longest path to item j where a path to item j is a sequence of the form $i_0 \to i_1 \to \ldots \to i_k \to \ldots \to i_K$ with $i_K = j$, $a_{i_{(k-1)}i_k} \ge 1$ for all $k \in \{1,...,K\}$ and $a_{hi_0} = 0$ for all items $h \in \{1,...,J\}$ and a distance of v_k between item i_k and $i_{(k+1)}$ for all $k \in \{0,...,K\text{-}1\}$.

(d) In the case of scarce machine capacity the sum of capacity units needed to produce one item j and all its predecessing components (denoted as cap_j) can be used as an indication of infeasibility when compared to the available capacity $AC_t = \sum_{\tau=1}^{t} C_\tau$.

A formal definition of the regret r_{jt} not to schedule an item j in period t can be given as follows (we can thing of other definitions which would be out of the scope of this paper): Let y be the item that is scheduled at the beginning of period t+1 (assume y = 0 if t = T).

Case 1: $CD_{jt} > 0$ and $j \neq y$ and $\sigma [(j , t)]$ = alive.

$$r_{jt} = \gamma_1 \frac{h_j CD_{jt}}{\max \{ s_i \mid \text{all items i} \}} \quad \text{(a)}$$

$$- \gamma_2 \frac{s_j}{\max \{ s_i \mid \text{all items i} \}} \quad \text{(b)}$$

$$+ \gamma_3 \frac{dep_j}{t - dep_j} \quad \text{(c)}$$

$$+ \gamma_4 \frac{CD_{jt}\, cap_j}{AC_t} \quad \text{(d)}$$

Case 2: $CD_{jt} > 0$ and $j = y$ and $\sigma [(j , t)]$ = alive.

Just drop (b) in case 1.

Case 3: $CD_{jt} = 0$ or $\sigma [(j , t)]$ = dead.

$$r_{jt} = -\infty$$

The real-valued parameters $\gamma_1,...,\gamma_4$ adjust the influence of the four criteria where $0 \leq \gamma_1,...,\gamma_4 \leq 1$ and $\sum_{i=1}^{4} \gamma_i = 1$ holds. To gain a better control over the influence of the differences between the item specific regret measures, a modified regret measure is introduced:

$$\rho_{jt} = \begin{cases} 0 & \text{if } r_{jt} = -\infty \\ (r_{jt} - \min \{ r_{it} \mid \text{items i with } r_{it} > -\infty \} + \varepsilon)^{\delta} & \text{otherwise} \end{cases}$$

This modified measure assigns a positive regret value to any item that may be selected for production. The small positive offset ε guarantees that all such items do indeed have a positive regret value. The exponent δ should by chosen as a non-negative value to amplify ($\delta > 1$) or to smoothen ($0 \leq \delta < 1$) the differences of the unmodified regrets.

Now that we have defined all ingredients of a hybrid cellular automaton based heuristic we can summarize the specification:

Step 1: *Initialize the cellular automaton.*

Step 2: *Choose the control parameters* $\gamma_1, ..., \gamma_4$, ε *and* δ *at random.*

Step 3: *(Try to) Generate a production plan that fits to the current setup state pattern.*

Step 4: *Evaluate the new production plan and memorize it if it improves the best one found so far.*

Step 5: *Choose the state-transition probabilities* $\pi_{a1}, ..., \pi_{a3}, \pi_{r1}, ..., \pi_{r3}, \mu_{a1}, ..., \mu_{a3}, \mu_{r1}, ..., \mu_{r3}$ *at random.*

Step 6: *Compute a new population of cells.*

Step 7: *Go to step 2 until a predefined number of iterations is performed.*

As stated above, how to choose the parameters in steps 2 and 5 is not clear (it may be specific to the problem instance to solve which values perform well) and is therefor done at random. The choice of items to be scheduled (step 3) can be done either deterministically by selecting the item with the largest modified regret (no item is to be scheduled if all regret values are zero) or stochastically with a probability proportional to the regret value.

6 Computational Study

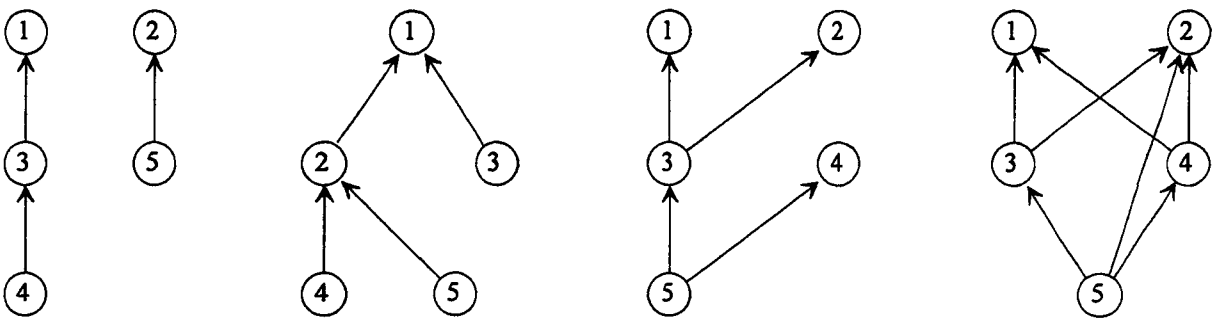
To study the presented heuristic we solved a total of 432 PLSP-instances. The results are compared to those that were achieved with a randomized regret based heuristic [Kimms 1993] which is the best one that we know of so far to solve multi-level, single-machine lot sizing and scheduling problems. Three different problem sizes are investigated:

(1) 5 items and 10 periods of time,

(2) 5 items and 25 periods of time, and

(3) 10 items and 25 periods of time.

The problem categories are a mixture of problem instances which are constructed on the basis of 4 different product structures with 5 items (see below). These structures are combined with different data sets, i.e. they differ with respect to demand patterns, item specific setup and holding costs, "gozinto"-factors, and capacity constraints.



Linear Structure Assembly Structure Divergent Structure General Structure

The probabilities $\pi_{a1}, ..., \pi_{a3}, \pi_{r1}, ..., \pi_{r3}, \mu_{a1}, ..., \mu_{a3}, \mu_{r1}, ..., \mu_{r3}$ as well as the parameters $\gamma_1, ..., \gamma_4$ are chosen from the interval [0 , 1] with a uniform distribution. The control parameters ε and δ of the randomized regret based heuristic are chosen at random from the intervals [0.0001 , 0.1] and [0 , 10], respectively.

In all cases we will provide the deviation of the result that was computed by the cellular automaton based heuristic from the result of the randomized regret based heuristic because most of the problems are to large to be solved with an exact standard MIP-solver. The deviation is defined as

$$\text{deviation} = 100 * \frac{F^*_H - F^*_{RR}}{F^*_{RR}}$$

where F^*_{RR} denotes the objective function value computed by the randomized regret based heuristic (RR) and F^*_H denotes the objective function value computed by the hybrid cellular automaton based heuristic (H). We consider a version of the cellular automaton based heuristic and the randomized regret based heuristic, respectively, in which an item is selected for production in a probabilistic way proportional to its regret measure. A negative value of the deviation indicates that the result of the randomized regret based procedure is improved.

For each problem instance a total of 1000 iterations are executed in order to find a feasible production plan. Every instance that we tested has at least one feasible solution. Figures 2, 3 and 4 give an overview of the results that were achieved for problem sizes (1), (2) and (3), respectively.

Figure 2: Hybrid heuristic vs. regret-based heuristic, problem size (1)

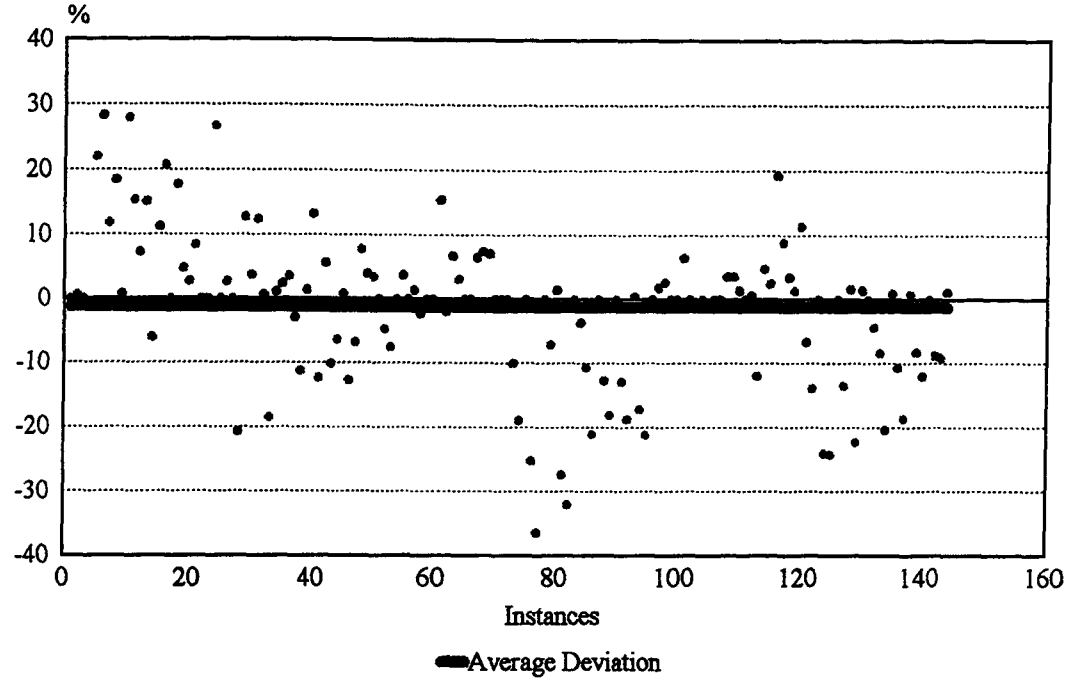
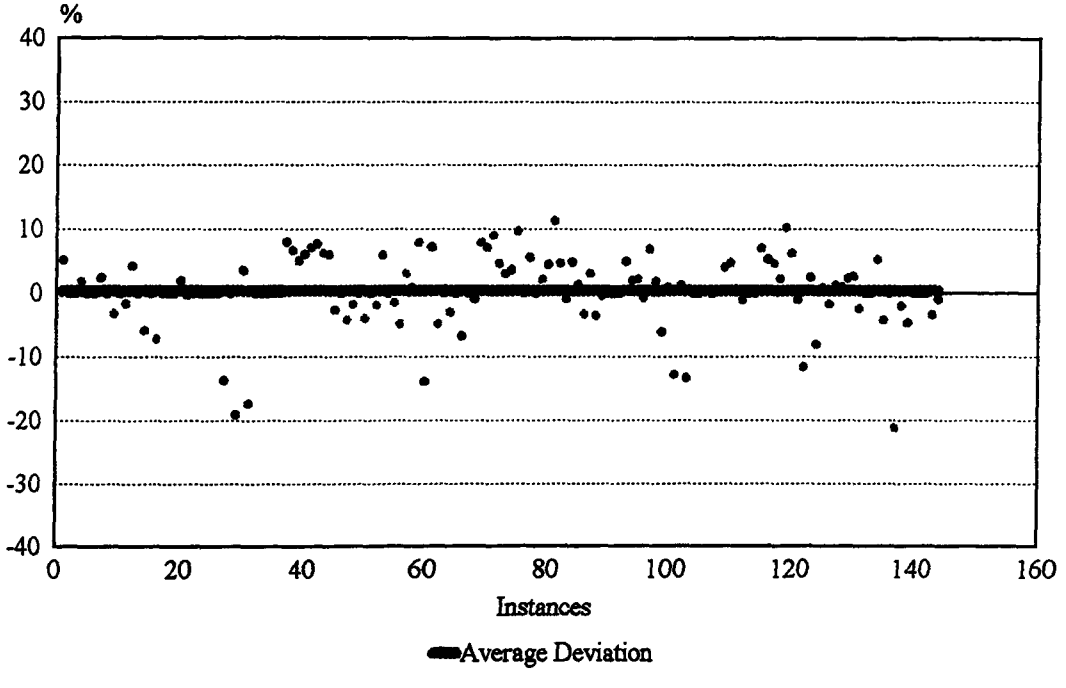
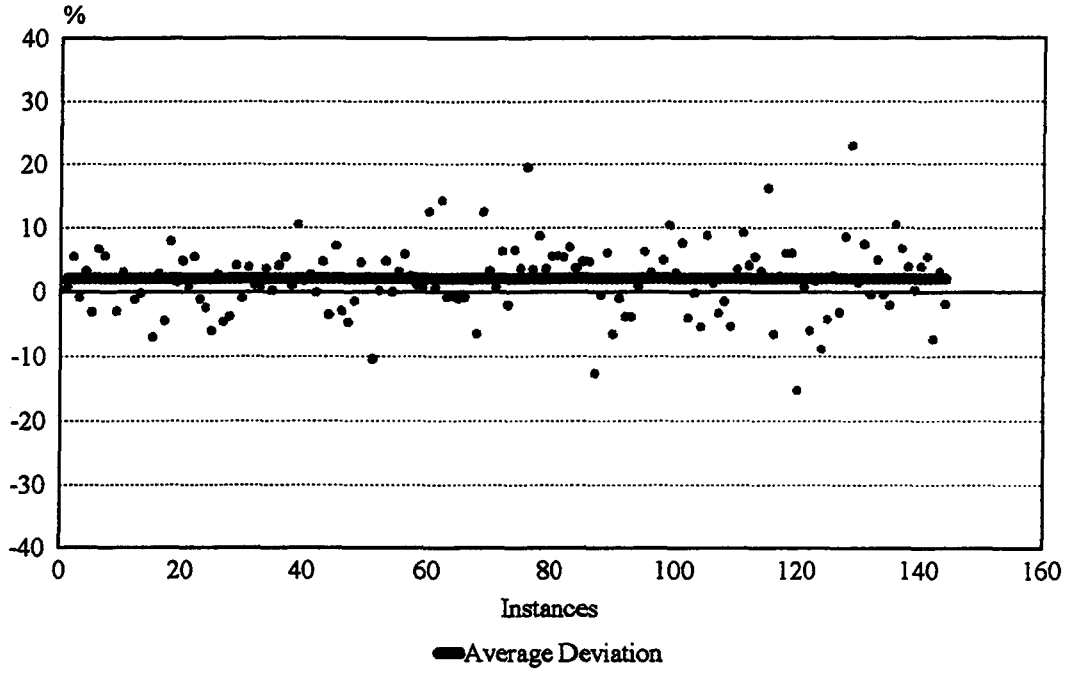


Figure 3: Hybrid heuristic vs. regret-based heuristic, problem size (2)



The results of the randomized regret based heuristic were slightly improved (i.e. -1.30 % in the average) in the case of size (1) problem instances, but in the case of size (2) and size (3) instances the results of the hybrid cellular automaton based heuristic were slightly worse (i.e. 0.30 % and 2.04 %, respectively, in the average). In terms of runtime the regret based heuristic outperforms the hybrid heuristic by a factor up to 7 on a 486 PC with 25 MHz. As it could have been expected in advance, the hybrid heuristic is dominated when we consider the infeasibility ratio, i.e. the number of feasible production plans that could be found during 1000 iterations.

Figure 4: Hybrid heuristic vs. regret-based heuristic, problem size (3)



Employing the cellular automaton based heuristic seems to be promising if it does not come to hand by the regret measure which item to schedule at next. This tends to be the case in product structures with many items because in such cases it may happen that few items have a large regret and many items have a low regret value. Due to the large number of items with a low regret it is therefor more probable to choose an item with a low regret instead of an item with a high regret value which usually should be the case. The cellular automaton approach reduces the number of items that are considered and thereby smoothes this effect. Moreover, the masking mechanism of the cellular automaton based heuristic may enforce a splitting of lots which may be necessary to find a good (or even feasible) solution.

7 Conclusion and Future Work

In this paper we presented the idea of cellular automata and sketched out that genetic algorithms are a special class of these. As an example we showed how this approach can be applied to multi-level, single-machine lot sizing and scheduling problems. Although the computational study brought out that in this particular case a cellular automaton based heuristic in general does not improve the results when compared to another good heuristic, it turned out that the results are not that bad in the average. It was shown that certain aspects of a problem can neatly been modelled (e.g. the fact that production often lasts several periods and that at most two items are produced within each period of time).

In general, one cannot declare the cellular automaton based approach to be superior to other techniques or vice versa. All one could say is that in some cases a cellular automaton will fit more easily to model a certain aspect of a problem, and in some cases it won't.

Future work should find out if and how cellular automata fit to other (integer and combinatorial) optimization problems (not only in the field of production planning). Heuristics should be developed and for

instance be compared to genetic algorithms, tabu search methods and simulated annealing approaches of which we know that they lead to fairly good results.

Acknowledgement

Prof. Dr. Andreas Drexl got not tired to read this paper and gave many helpful suggestions.

References

Collins, N. E., Eglese, R. W., Golden, B. L., (1988), Simulated Annealing - An Annotated Bibliography, American Journal of Mathematical and Management Sciences, Vol. 8, pp. 209-307

Dorndorf, U., Pesch, E., (1992), Evolution Based Learning in a Job Shop Scheduling Environment, Research Memorandum, Limburg University, Maastricht, Computers and Operations Research, to appear

Drexl, A., (1991), Scheduling of Project Networks by Job Assignment, Management Science, Vol. 37, pp. 1590-1602

Drexl, A., Haase, K., (1992), A New Type of Model for Multi-Item Capacitated Dynamic Lotsizing and Scheduling, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 286

Faigle, U., Kern, W., (1992), Some Convergence Results for Probabilistic Tabu Search, ORSA Journal on Computing, Vol. 4, No. 1, pp. 32-37

Gardner, M., (1970), The Fantastic Combinations of Conway's New Solitaire Game "Life", Scientific American, Vol. 223, No. 4, pp. 120-123

Glover, F., (1989), Tabu Search - Part I, ORSA Journal on Computing, Vol. 1, pp. 190-206

Glover, F., (1990a), Tabu Search - Part II, ORSA Journal on Computing, Vol. 2, pp. 4-32

Glover, F., (1990b), Tabu Search: A Tutorial, Interfaces, Vol. 20, pp. 74-94

Goldberg, D. E., (1989), Genetic Algorithms in Search, Optimization and Machine Learning, Reading, Addison-Wesley

Haase, K., (1993), Lotsizing and Scheduling for Production Planning, Ph.D. thesis, University of Kiel

Hertz, A., de Werra, D., (1990), The Tabu Search Metaheuristic: How we used it, Annals of Mathematics and Artificial Intelligence, Vol. 1, pp. 111-121

Holland, J. H., (1975), Adaption in Natural and Artificial Systems, Ann Harbor, University of Michigan Press

Johnson, D. S., Aragon, C. R., McGeoch, L. A., Schevon, C., (1989), Optimization by Simulated Annealing: An Experimental Evaluation: Part I - Graph Partitioning, Operations Research, Vol. 37, pp. 865-892

Johnson, D. S., Aragon, C. R., McGeoch, L. A., Schevon, C., (1991), Optimization by Simulated Annealing: An Experimental Evaluation: Part II - Graph Coloring and Number Partitioning, Operations Research, Vol. 39, pp. 378-406

Kimms, A., (1993), Multi-Level, Single-Machine Lot Sizing and Scheduling (with Initial Inventory), Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 329

Liepins, G. E., Hilliard, M. R., (1989), Genetic Algorithms: Foundations and Applications, Annals of Operations Research, Vol. 21, pp. 31-58

Mühlenbein, H., Gorges-Schleuter, M., Krämer, O., (1988), Evolution Algorithms in Combinatorial Optimization, Parallel Computing, Vol. 7, pp. 65-85

Nemhauser, G. L., Wolsey, L. A., (1988), Integer and Combinatorial Optimization, New York, Wiley

Sinclair, M., (1993), Comparison of the Performance of Modern Heuristics for Combinatorial Optimization on Real Data, Computers and Operations Research, Vol. 20, No. 7, pp. 687-695

Toffoli, T., Margolus, N., (1988), Cellular Automata Machines, Cambridge, MIT Press

Wolfram, S., (1986), Theory and Applications of Cellular Automata, World Scientific

Zanakis, S. H., Evans, J. R., Vazacopoulos, A. A., (1989), Heuristic Methods and Applications: A Categorized Survey, European Journal of Operational Research, Vol. 43, pp. 88-110