Production Sequencing as Negotiation

Michael Wooldridge[†] Stefan Bussmann[‡] Marcus Klosterberg[‡]

[†] Department of Computing, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, United Kingdom
M.Wooldridge@doc.mmu.ac.uk

[‡] Daimler-Benz AG, Forschung und Technik, Alt-Moabit 96a, 10559 Berlin, Germany
{bussmann, klosterb}@DBresearch-berlin.de

Abstract

The production sequencing problem involves a factory generating a product sequence such that when processed, the sequence will both satisfy current orders, and minimize overall costs. In this paper, we argue that production sequencing may fruitfully be considered as a negotiation problem, in which production cells within a factory negotiate over product sequences in order to fairly distribute costs. We begin by describing and formally defining the production sequencing problem; we then investigate its complexity, and present an analysis of it using the tools of game and negotiation theory. We then define a negotiation algorithm for production sequencing, and discuss what assumptions and simplifications must be made in order to make this algorithm suitable for implementation. We conclude by discussing issues for future work

1 Introduction

Multi-agent technology has, in the past decade, moved from being a somewhat obscure relation of mainstream AI and computer science to being what is widely accepted as one of the most vibrant and exciting research areas in computing generally. As with all new software technologies, the ratio of speculation and hype to field tested, implemented systems is perhaps somewhat high, but there are, nevertheless, some documented multi-agent applications. Example domains include power systems management [13], air-traffic control [11], particle accelerator control [2], telecommunications network management [12], spacecraft control [10], computer integrated manufacturing [7], job shop scheduling [5], and steel coil processing control [4]. In this paper, we present a multi-agent solution for an industrial problem that is both common and important: *factory production sequencing*. Briefly, this problem involves a factory deciding upon the sequence in which to process products, so as to both satisfy current orders, and minimize overall factory costs.

The main import of this paper is that production sequencing may be treated as a *multi-agent negotiation* problem, in which production cells within a factory negotiate over product sequences in order to distribute costs fairly, and hence minimize total factory costs. There are a number of advantages to such an approach, chief among them being *modularity* and *flexibility*. We give a formal definition of the problem domain, discuss its complexity (in particular, we show that the problem is, in general, NP-complete), and present an analysis of it using the tools of game and negotiation theory. We then describe a negotiation mechanism for the production sequencing problem, and discuss what assumptions and simplifications might be required in order to implement this mechanism. The paper concludes with a discussion of future research issues. We begin, in the following sub-sections, by presenting an overview of the problem domain, and a rationale for treating the problem as one of negotiation.

1.1 The Problem Domain: An Overview

Within a typical manufacturing plant, there will be a number of *production cells*, that perform different tasks. Examples of such cells include paint spraying, cleaning, buffering of products, assembly, and so on. Every week, (in some factories, every day), it is necessary to plan how these cells are to be used to fulfill product orders. Such a plan is known as a *production sequence*. It defines the order in which products will pass through the various factory cells during the manufacturing process.

Some production sequences cost more than others to process. To see why, consider the following example. A factory has a cell that does paint spraying; products may be sprayed in any number of colours, but once the cell has sprayed a product some particular colour, changing to another colour requires that the cell be cleaned, nozzles and perhaps paint tanks be replaced, and so on. This takes time and money. In this example, it will be cheaper, (from the point of view of the paint spraying cell), to process the production sequence $\langle red, red, green\rangle$ than it would be to process the sequence $\langle red, green, red\rangle$.

The situation is complicated by the fact that different cells have different requirements for product sequences. Extending the simple paint spraying example given above, imagine that the factory contains another production cell, for assembly. Products that pass through the assembly cell may be fitted with either a petrol engine or a diesel engine. Suppose that one red car required a petrol engine, the other red car required a diesel engine, and the green car required a petrol engine. From the point of view of the assembly cell, therefore, it might be cheaper to process the sequence

$$\langle\langle red, petrol\rangle, \langle green, petrol\rangle, \langle red, diesel\rangle\rangle$$

than the sequence

$$\langle\langle red, petrol\rangle, \langle red, diesel\rangle, \langle green, petrol\rangle\rangle$$

that would be preferred by the paint spraying cell[1]. It is not difficult to see that there is in some sense potential for conflicts between the production cells within a factory. Finding the *cheapest* production sequence with respect to the *whole* factory (i.e., all production cells) is what we call the *production sequencing problem*. Solving the problem is something of a balancing act, which may end up with some cells having to accept a production sequence that is, from their point of view, sub-optimal, in order to minimize the overall factory cost.

1.2 Production Sequencing as Negotiation

In this paper, we propose and formally analyze an approach to production sequencing in which finding the cheapest sequence that satisfies a particular order becomes a *multi-agent negotiation*

[1]The sequence $\langle\langle green, petrol\rangle, \langle red, petrol\rangle, \langle red, diesel\rangle\rangle$ would be a solution that kept both cells happy.

problem. The basic idea of this approach is very simple. Recall that a factory is made up of a number of cells. Each of these cells has associated with it a cost function, which characterizes the cost to that cell of processing any production sequence. These cost functions implicitly define preferences over production sequences. (Recall the paint spraying example, above.) In multi-agent production sequencing, we make each cell an agent, and get them to *negotiate* a production sequence. Each agent attempts to improve its *utility* by reducing costs: it will argue for production sequences that reduce these costs. Eventually, if we have designed our agents right, they will come to agree on a sequence that is, in some sense, optimal. It may not be a *globally* optimal solution, but at least a reasonable one.

There are several potential advantages to the multi-agent production sequencing approach:

- *Modularity*. Ultimately, production sequencing is an optimization problem. If we had available a complete mathematical model of a factory, then we might, in principle, use optimization techniques to solve the production sequencing problem for that factory. But such models are *not* available, and attempting to solve the problem in this way is entirely unrealistic: the production process in any real factory is simply too complex. The multi-agent approach provides us with an obvious, familiar tool to help manage this complexity: modularity. It allows us to break the overall problem down into sub-problems of a more manageable size, in a way that accurately reflects the structure and operation of the factory. The representation of conflicts and other interactions between these modular units then becomes a realistic possibility.

- *Adequate Modeling*. Monolithic, centralized solutions to the production sequencing problem could not, in general, serve as adequate *models* of the production sequencing process. This is because they do not represent the organizational structure inherent within a factory. Multi-agent approaches offer a clear advantage in this respect, as they enable us to represent the real-world organizational entities (i.e., production cells) that take part in the generation of a production sequence.

- *Flexibility*. In our discussions with production managers, we have found it stressed over and over again that factories are not static entities. They are in a state of almost continual change, with new equipment, products, and working practices being frequently introduced. Every time such a change occurs, the costing profile of the factory changes accordingly. Monolithic, centralized systems are simply not adaptable enough to deal with such a dynamic domain. In contrast, a multi-agent approach, in which knowledge about costs is distributed among agents corresponding to production cells, would allow changes in cost information to be easily incorporated.

- *Parallelism*. Finally, since a multi-agent approach is inherently distributed, there is potential for *parallelism* in the solution. Such parallelism might improve the speed with which a solution is found.

2 Formal Analysis of the Problem Domain

In this section, we formally define the production sequencing problem, and present an analysis of it using the tools of game and negotiation theory. We begin by defining our notation.

2.1 Notation

We use standard set theoretic notation where possible, augmented as follows. The cardinality of a set S is denoted $\#S$. The set of *bags* (multisets) over some set S is denoted by bag S. The number of times that an element x occurs in a bag b is denoted $x\#b$. The set of *sequences* over some set S is denoted seq S. Finally, if S is an arbitrary (non-empty) set, $\sigma \in$ seq S, and $x \in S$, then $occurs(x, \sigma)$ denotes the number of times x occurs in σ.

2.2 Formal Definition of the Domain

The production sequencing problem may be formalized as follows. First, let the set of all possible *product types* be P. We generally abbreviate 'product types' to 'products', and use p (with decorations: $p', p_1, \ldots$) to stand for members of P. An *order* is a bag (multiset) of products. Let $O =$ bag P be the set of all orders. We use o (with decorations: o', $o_1, \ldots$) to stand for members of O. If $o \in O$, then let $\#o$ denote the total size of the order o — the total number of all products in o. Thus

$$\#o = \sum_{p \in P} p\#o.$$

Let $\mathit{diff}(o) \subseteq P$ denote the set of *different* products in o, i.e., $\mathit{diff}(o) = \{p \mid p \in P \text{ and } p\#o > 0\}$. If $\#\mathit{diff}(o) = \#o$, then every product in o occurs only once, i.e., o contains no duplicate products.

A *production sequence* is simply a sequence over P; let the set of all production sequences be S, i.e., $S =$ seq P. We use σ (with decorations: $\sigma', \sigma_1, \ldots$) to stand for members of S. A production sequence $\sigma \in S$ is said to *satisfy* an order $o \in O$, (notation: $\mathit{satisfies}(\sigma, o)$) iff $\forall p \in P$, we have $p\#o = occurs(p, \sigma)$, i.e., the sequence σ contains exactly as many of product p as are contained in o. If $o \in O$, then let $sat(o) \subseteq S$ denote the set of production sequences that satisfy order o, i.e., $sat(o) = \{\sigma \mid \sigma \in S \text{ and } \mathit{satisfies}(\sigma, o)\}$.

A *cost function*, c, has the signature $c : S \to I\!R$, i.e., it takes a production sequence and returns a real number which represents the cost of processing that sequence. We generally associate cost functions with production cells, in which case the cost function represents the cost *to that cell* of processing the sequence. A *factory*, f, is defined to be an indexed set of cost functions $\{c_i\}$, one for each cell i in the factory. Given a factory $f = \{c_i\}$ it is possible to derive a *global cost function*, $c_f : S \to I\!R$, that gives the cost to the whole factory of processing a particular production sequence:

$$c_f(\sigma) \overset{\text{def}}{=} \sum_{c_i \in f} c_i(\sigma)$$

Clearly, a production sequence $\sigma \in S$ will be globally preferred (i.e., preferred from the point of view of the factory) to a production sequence $\sigma' \in S$ with respect to factory f iff $c_f(\sigma) < c_f(\sigma')$. We can now formally state the production sequencing problem.

> THE PRODUCTION SEQUENCING PROBLEM (PSP): Given an order $o \in O$ and a factory f, find a production sequence that both satisfies o and is *globally optimal* with respect to c_f, i.e., a sequence $\sigma \in sat(o)$ that has the property that $\nexists \sigma' \in sat(o)$ such that $c_f(\sigma') < c_f(\sigma)$.

2.3 Complexity

A naive solution to the PSP would involve exhaustively searching the set $sat(o)$ of all production sequences that satisfy order $o \in O$, in order to find the one with minimum cost. It is not difficult to see that the size of the search space, $\#sat(o)$, will be exponential in the number of different products in o. We present an equation that precisely defines the size of $sat(o)$. Let $t_i = p_i\#o$, for all $p_i \in \mathit{diff}(o)$. Thus t_i represents the number of different products of type p_i contained in o. Let $n = \#o$; thus n represents the total number of all products in o, i.e., the length of any production sequence that satisfies o. Let $k = \#\mathit{diff}(o)$; thus k is the number of different products in o. Then $\#sat(o)$, the size of the search space, is given by:

$$\#sat(o) = \begin{pmatrix} n \\ t_1 \end{pmatrix} \begin{pmatrix} n - t_1 \\ t_2 \end{pmatrix} \begin{pmatrix} n - (t_1 + t_2) \\ t_3 \end{pmatrix} \cdots \begin{pmatrix} n - \Sigma_{j=1}^{k-1} t_j \\ t_k \end{pmatrix} \tag{1}$$

where

$$\begin{pmatrix} n \\ t \end{pmatrix} = \frac{n!}{(n-t)!t!}.$$

Since all the variables on the right hand side are independent, we can do no simplification. However, for certain cases, it *is* possible to simplify. Consider the case where the same number of each product type is required. In this case, $\forall p, p' \in P$, if $p\#o > 0$ and $p'\#o > 0$, then $p\#o = p'\#o$. Let us write t for the number of products required of each type. Clearly, $n = kt$. In this case, equation (1) reduces to

$$\#sat(o) = \frac{n!}{t!^{(n/t)}} \tag{2}$$

In the case where $n = k$, (i.e., the order contains no duplicate products), equation (2) reduces to $\#sat(o) = n!$ since $t = 1$. In the case where $k = 1$ (i.e., there is only one product type), then $\#sat(o) = 1$, irrespective of the size of n.

NP-Completeness

We will now prove that for an important subset of PSP problems, finding an optimal sequence is NP-complete (and hence, if P $\neq$ NP, then the PSP cannot in general be solved in better than exponential time). We pre-suppose some familiarity the concept of NP-complete problems [6].

The sub-class of the PSP that we consider is that in which the cost function c_f for a factory f is *stepped*. By this, we mean that the *global* cost of a production sequence is determined from *local* properties. Formally, a cost function c_f is stepped iff there exists some function $\bar{c_f} : P \times P \to I\!R$ such that

$$c_f(\sigma) = \sum_{i=1}^{(\#\sigma)-1} \bar{c_f}(\sigma(i), \sigma(i+1)).$$

Given a stepped cost function c_f, we generally use $\bar{c_f}$ to denote the function with signature $P \times P \to I\!R$ that corresponds to it. By the term 'stepped PSP', we mean the PSP with stepped cost functions; (we recast the problem as a decision problem, in the standard way).

> THE STEPPED PSP: Given an order $o \in O$ (over P), a stepped cost function $\bar{c_f} : P \times P \to I\!R$, and a value $k \in I\!R$, determine whether or not there is any production sequence $\sigma \in sat(o)$ with total cost less than k.

If we know that c_f is stepped, then we can recover the corresponding function $\bar{c_f}$ from c_f in time $O(\#P^2)$. The key result of this section is as follows.

Theorem 1 *The stepped PSP is NP-complete.*

PROOF: (Outline) Membership of NP is easy; completeness is by a polynomial reduction to the travelling salesman problem. □

2.4 An Example

To illustrate the ideas presented above, we present a simple example. A car factory f consists of three production cells: cell 1 does product assembly, cell 2 does paint spraying, and cell 3 does quality control. Cars have three relevant attributes: colour (red or green), engine type (petrol or diesel), and drive side (left or right). There are thus a total of 8 different possible products. The cost functions for each cell are defined as follows (notice that the factory cost function c_f induced by these cell cost functions is *stepped*, in the sense described above):

c_1 : It costs cell 1 a total of 5 units to process each product, plus 5 units for every change of engine type, and an additional 5 units for every change of drive side.

c_2 : It costs cell 2 a total of 5 units to process every product, plus 5 units for every colour change.

c_3 : It costs cell 3 a total of 5 units to process every product, plus 5 units for every change of engine type.

Define products p_1, p_2, and p_3 as follows:

$$p_1 = \langle red, petrol, left\rangle \qquad p_2 = \langle red, diesel, right\rangle \qquad p_3 = \langle green, diesel, left\rangle.$$

Now consider an order that requires $2 \times p_1$, $1 \times p_2$, and $1 \times p_3$. Equation (1) tells us that there are exactly 12 production sequences that satisfy this order. These production sequences, together with the corresponding cost for each cell and the cumulative factory cost, are summarized in Table 1.

It is not difficult to see from Table 1 that σ_1, σ_2, and σ_{12} are the only globally optimal solutions. The best sequence for agent (cell) 1 is σ_2; for agent 2, the best sequences are σ_1, σ_3, σ_7, σ_{10}, σ_{11}, and σ_{12}; finally, for agent 3, the best sequences are σ_1, σ_2, σ_9, and σ_{12}. There is no globally optimal solution that is locally optimal for all agents.

2.5 Game Theoretic Analysis

We now present an analysis of the PSP, using the tools of game and negotiation theory [3, 8]. The idea is that since we intend to use negotiation to attack a particular problem — the PSP — we should first determine how well this problem maps into standard negotiation theoretic domains. An obvious place to start is [8], where three types of domain are defined: *task oriented domains* [8, p30]; *state oriented domains* [8, p90]; and *worth oriented domains* [8, p155]. The details of these domains are not relevant to our discussion: the important point is that the PSP does not correspond to any of them. To see why, we need to examine the assumptions that underpin these domains. In each case, we find it is assumed that agents may *choose* whether or not to cooperate (i.e., they are autonomous). If there is no cooperative solution that makes them better off than they would be on

		$c_1(\sigma_n)$	$c_2(\sigma_n)$	$c_3(\sigma_n)$	$c_f(\sigma_n)$
σ_1	$\langle p_1, p_1, p_2, p_3 \rangle$	30	25	25	80
σ_2	$\langle p_1, p_1, p_3, p_2 \rangle$	25	30	25	80
σ_3	$\langle p_1, p_2, p_1, p_3 \rangle$	40	25	35	100
σ_4	$\langle p_1, p_2, p_3, p_1 \rangle$	35	30	30	95
σ_5	$\langle p_1, p_3, p_1, p_2 \rangle$	30	30	35	95
σ_6	$\langle p_1, p_3, p_2, p_1 \rangle$	35	30	30	95
σ_7	$\langle p_2, p_1, p_1, p_3 \rangle$	35	25	30	90
σ_8	$\langle p_2, p_1, p_3, p_1 \rangle$	40	30	35	105
σ_9	$\langle p_2, p_3, p_1, p_1 \rangle$	30	30	25	85
σ_{10}	$\langle p_3, p_1, p_1, p_2 \rangle$	30	25	30	85
σ_{11}	$\langle p_3, p_1, p_2, p_1 \rangle$	35	25	35	95
σ_{12}	$\langle p_3, p_2, p_1, p_1 \rangle$	30	25	25	80

Table 1: Production Sequences and Their Costs

their own (i.e., there is no *individual rational* joint solution), then cooperation will not occur. This is because agents are *utility maximizers*: they will always select the course of action that gives them personally the highest payoff, irrespective of the consequences for the society to which they belong.

The assumption that agents are utility maximizers does not correspond to the PSP domain. Our agents have no choice about whether to cooperate, and are not, therefore, fully autonomous. Conceptually, our factory agents share a *common goal* of finding a negotiation sequence that minimizes overall costs. However, if we are to use negotiation theoretic techniques for the PSP, then we are required to find concepts in our domain that correspond to negotiation theoretic notions such as utility, pareto optimality, and so on. That is what we aim to do in this sub-section. First, we collect together some of the concepts presented above, and define what we mean by a *multi-agent production sequencing domain*.

Definition 1 *A* multi-agent production sequencing domain *(MPSD) is a structure*

$$\langle P, O, S, Ag, f \rangle$$

where:

- P *is a set of* products*;*
- $O = \text{bag } P$ *is the set of* orders *over* P*;*
- $S = \text{seq } P$ *is the set of* production sequences *over* P*;*
- $Ag = \{1, \ldots, l\}$ *is a set of* agents*, (corresponding to production cells);*
- $f = \{c_i\}$ *is a factory, i.e., an indexed set of cost functions, one for each agent (cell)* $i \in Ag$*.*

Next, we define the notion of an *encounter*. In the standard game theoretic sense, an encounter is a particular instantiation of a domain, together with a situation in which agents have goals to achieve.

Definition 2 *An* encounter *in an MPSD $\langle P, O, S, Ag, f\rangle$ is simply an order, i.e., a member of the set O.*

Since we can, in principle, compute the cost to any agent of any production sequence, and there will be a finite (if somewhat large) number of production sequences that satisfy a particular encounter, we can compute the *worst* that a particular agent can do with respect to an encounter.

Definition 3 *If i is an agent in an MPSD $\langle P, O, S, Ag, f\rangle$, then the* worst *that i can do in encounter $o \in O$ (notation: $worst_i(o)$) is defined $worst_i(o) = \max\{c_i(\sigma) \mid \sigma \in sat(o)\}$.*

The best that i can do with respect to order o is denoted $best_i(o)$; the formal definition of $best_i(o)$ is similar to that of $worst_i(o)$, and is therefore omitted. Next, we define the notion of a *deal*. The rationale behind this terminology will become clear when we discuss negotiation mechanisms in section 3.

Definition 4 *A* deal *in an MPSD $\langle P, O, S, Ag, f\rangle$ with respect to an encounter $o \in O$ is a production sequence that satisfies o, i.e., a member of the set $sat(o)$.*

Next, we can define the *utility* of a deal with respect to some encounter.

Definition 5 *The* utility *of a deal σ for some agent $i \in Ag$ with respect to an encounter o (notation: $utility_i^o(\sigma)$) is defined to be the difference between the worst that i can do in o, and the cost to i of σ: $utility_i^o(\sigma) = worst_i(o) - c_i(\sigma)$.*

Since o is generally understood from context, we usually omit reference to it.

Definition 6 *Let σ and σ' be deals for some encounter in an MPSD $\langle P, O, S, Ag, f\rangle$. Then:*

- *σ is said to* weakly dominate *σ' from the point of view of group $g \subseteq Ag$, (notation: $\sigma \succeq_g \sigma'$) iff $\forall i \in g$, $utility_i(\sigma) \geq utility_i(\sigma')$, i.e., if every agent in g does at least as well in σ as it does in σ';*
- *σ is said to* dominate *σ' from the point of view of group $g \subseteq Ag$ (notation: $\sigma \succ_g \sigma'$) iff $\sigma \succeq_g \sigma'$ and $\exists i \in g$ for whom $utility_i(\sigma) > utility_i(\sigma')$, i.e., if every agent in g does at least as well in σ as in σ', and at least one agent does better in σ than in σ'.*

(When referring to the set of all agents, we simply write $\succeq$ or $\succ$, rather than $\succeq_{Ag}$ or $\succ_{Ag}$.) We can now define the well-known game theoretic notion of *pareto optimality* for our domain [3].

Definition 7 *A deal σ is said to be* pareto optimal *iff there exists no other deal σ' such that $\sigma' \succ \sigma$.*

Pareto optimality is an important concept in game theory. To see why, suppose a group of autonomous (utility maximizing) agents are attempting to negotiate, and they come across a deal that is pareto optimal. Then they may well have to fix upon this solution, because, by definition, choosing another solution would make some agent worse off — and no utility maximizing agent would agree to being made worse off. Thus, negotiation algorithms are often designed to generate pareto optimal solutions. But unfortunately for us, such solutions are not necessarily ideal:

Theorem 2 *With respect to the PSP, (1) global optimality implies pareto optimality, but (2) pareto optimality does not imply global optimality.*

PROOF: For (1), suppose not. Then there exists some globally optimal deal σ that is not pareto optimal. If σ is not pareto optimal, then there exists some other deal σ' in which every agent does at least as well as in σ, and some agent actually does better. But this implies that $c_f(\sigma') < c_f(\sigma)$, in which case σ is not globally optimal. But this is a contradiction, so the assumption must be false. For (2), the intuition is easy. Pareto optimality says that there is no other deal that increases the utility of one agent without reducing the utility of at least one other agent. But to get a *globally* optimal solution, one agent might have to accept a worse deal, in order to maximize the sum of the utilities, and hence minimize c_f. Here is a simple example to illustrate this. Suppose we have three agents, 1, 2, and 3, and only two possible deals, σ_1 and σ_2, with cost functions defined thus:

$$\begin{array}{rclrclrcl} c_1(\sigma_1) & = & 10 & c_2(\sigma_1) & = & 10 & c_3(\sigma_1) & = & 10 \\ c_1(\sigma_2) & = & 15 & c_2(\sigma_2) & = & 5 & c_3(\sigma_2) & = & 5 \end{array}$$

In this example, $c_f(\sigma_1) = 30$ and $c_f(\sigma_2) = 25$, hence σ_2 is the globally optimal solution. But σ_1 is pareto optimal, since although σ_2 increases the utility of agents 2 and 3, it decreases the utility of agent 1. $\Box$

This theorem indicates that, ideally, we seek negotiation strategies that are optimal in a stronger sense than pareto optimality. Finally, we can define the *negotiation set* for an encounter. The negotiation set intuitively represents the ‘reasonable’ deals that an agent could propose during negotiation.

Definition 8 *The* negotiation set *for an encounter $o \in O$ in some MPSD $\langle P, O, S, Ag, f \rangle$ is denoted NS^o, and is defined to be the set of pareto optimal deals for o.*

(As above, we generally omit reference to o.) It would be ‘unreasonable’ of an agent to propose a deal that was not pareto optimal since by definition, the agent could have proposed another deal that made one agent better off without making any other agent worse off[2]. An obvious corollary of Theorem 2 is that the negotiation set for any encounter is guaranteed to be non-empty: there will always be a globally optimal solution, which, by Theorem 2, must be pareto optimal.

2.6 Back to the Example

Let us return to the example that we presented in section 2.4. We can analyze each of the solution sequences to this problem using the terminology and techniques just introduced. First, note that the worst that agent 1 can do is cost 40 (sequences σ_3 and σ_8). The worst that agent 2 can do is cost 30 (sequences σ_2, σ_4, σ_5, σ_6, σ_8, and σ_9). Finally, the worst that agent 3 can do is cost 35 (σ_3, σ_5, σ_8, and σ_{11}). Knowing the worst that an agent can do allows us to compute the utility for that agent of each sequence; this in turn allows us to determine whether or not a sequence is pareto optimal. These results are summarized in Table 2. The negotiation set, NS, thus consists of just 3 deals: σ_1, σ_2, and σ_{12}.

[2]Note that *individual rationality* has no meaning in our domain (cf. [8, p39]).

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}
$utility_1(\sigma_n)$	10	15	0	5	10	5	5	0	10	10	5	10
$utility_2(\sigma_n)$	5	0	5	0	0	0	5	0	0	5	5	5
$utility_3(\sigma_n)$	10	10	0	5	0	5	5	0	10	5	0	10
pareto optimal?	$\times$	$\times$										$\times$

Table 2: Properties of Production Sequences

3 Negotiation Mechanisms for the MPSD

Thus far, we have said very little about the *negotiation process* that might be used in the MPSD. In this section, we address this issue. We begin with an informal discussion of how the negotiation process might work. We then consider some desirable properties of a negotiation mechanism for the MPSD, and in section 3.2, we formally define a negotiation algorithm. However, this algorithm makes certain assumptions about agents that are unrealistic in practice. Therefore, in section 4, we consider the implementation aspects of the algorithm, and in particular, we suggest how it might be made suitable for implementation.

As we observed in section 1.2, the intuition behind our work is that the process of finding a production sequence might usefully be viewed as a negotiation problem, in which factory cells negotiate over production sequences in order to minimize their *local* costs, and hence, it is hoped, reduce *global*, or *factory* costs. But what, exactly, are the agents to negotiate *over*? What proposals are agents going to make during the negotiation process? There are at least two possibilities:

- agents negotiate over *individual products* in order to *incrementally develop* an optimal production sequence;
- agents negotiate over *entire production sequences* in order to find one that minimizes costs.

Let us briefly consider the first possibility. The idea here is that some agent will begin by proposing one particular product, which it desires to be the first product in the sequence. Other agents will then make counter offers, until agreement is somehow reached. The agents then move on to negotiate the second product in the sequence, and so on, until an entire sequence is developed that satisfies current orders. There are several obvious drawbacks to this approach. First, and perhaps most importantly, it requires that agents make very *local* decisions, with the ultimate aim of satisfying a *global* requirement [1, pp21–22]. It seems difficult to devise heuristics appropriate to the domain that might be used by an agent in order to meet this requirement. Secondly, this method does not correspond to our intuitions about how humans solve the problem. Typically, humans will start with an entire sequence that approximates to optimality, and then iteratively refine this sequence. For these reasons, we focus in this paper on the second approach, in which agents negotiate over entire production sequences. Of course, this approach has problems of its own: we discuss these in section 4.

3.1 Desiderata for an MPSD Negotiation Mechanism

As we noted in section 1.2, a multi-agent approach to production sequencing has several inherent advantages over a monolithic approach (e.g., flexibility, modularity). In addition, an ideal negotiation mechanism for the MPSD would satisfy certain other properties (cf. [8, pp20–22]):

Simplicity: We seek a negotiation mechanism that will minimize the time taken to reach agreement on a deal. Realistically, it may not be possible to find a tractable (polynomial time) negotiation mechanism for the MPSD. Perhaps the best we can hope for is an 'anytime' negotiation mechanism (cf. [9]): one that causes the agents to quickly find *some* (sub-optimal) solution, and will then monotonically improve solution quality, as long as the agents are allowed to continue negotiating.

Efficiency: The purpose of the PSP is to find a production sequence that minimizes, as far as possible, the cost to a factory of processing a particular order. This is a very real problem: even small improvements in day-to-day production sequencing lead to significant reductions in factory running costs. Ideally, we therefore seek a negotiation mechanism that will lead to agreement on a *globally optimal* deal. Realistically, it may be that we can do no better than, say, pareto optimality.

It is worth noting that, because of the nature of our domain, some of the attributes of negotiation as discussed in negotiation and game theory are not relevant for our purposes [8, pp20–21]. For example, a key concept in negotiation theory is *stability*, as represented in the property of *Nash equilibrium* [8, p190]. Two negotiation strategies s and s' are said to be in Nash equilibrium iff under the assumption that one agent is using s, another agent can do no better than use s', and vice versa. This property is relevant because of the standard game theoretic assumption that agents are utility maximizers: they each have their own goals, and will try to achieve these goals at the expense of other agents, if necessary. Hence an agent will always choose a strategy that maximizes its own utility. This notion is not relevant in the MPSD domain because our agents will all be designed to meet the *common* goal of reducing total factory costs. For the same reason, we can ignore the whole issue of deception [8, pp53–85]: our agents will not try to deceive other agents because, if they did so, they might improve their own utility at the expense of the factory.

3.2 A Negotiation Mechanism

In this section, we present a negotiation algorithm for the MPSD. This algorithm is an extended and adapted version of the *Monotonic Concession Protocol* [8], in which agents use a generalized version of the *Zeuthen strategy* [14]. We begin with a general overview of the mechanism, and then give a rigorous definition using the notation and terminology of previous sections.

Overview

The basic idea is quite simple. Negotiation proceeds in *rounds*, and on the first round, every agent takes an active part by proposing some deal. If a deal has been proposed that makes every agent happy, then negotiation ends successfully. Otherwise, negotiation proceeds to another round, in which some subset of the currently active agents must *concede*. For such conceding agents, there are three possibilities:

- the agent is able to propose a deal that represents a 'true' concession, in which case it does so;
- the agent is unable to make a 'true' concession, but is nevertheless able to make another proposal, in which case it does so; in this case, the agent will in some sense be 'backtracking';

- the agent has exhausted the set of all deals it could propose, in which case it withdraws, and plays no further part in negotiation.

Agents that do not concede in some round put forward the same deal on the next round. In this way, negotiation proceeds with agents conceding and possibly withdrawing, until finally, they find a deal upon which they agree. Intuitively, the agents involved in such negotiation are searching through the negotiation set NS, in an attempt to find a mutually acceptable deal. The aim is to find heuristics that guide the search to an efficient solution as quickly as possible. On examining the basic algorithm, as described above, it becomes apparent that there are three key questions to be answered:

- how is an agent to choose its *first proposal*, and any *backtrack proposal*?
- on any given round, *who should concede*?
- if an agent concedes, then *how much* should it concede?

In the sub-sections that follow, we present solutions to these three problems, and then give a formal statement of the entire negotiation mechanism.

What Should the First Proposal Be?

When negotiation begins, an agent must select from the negotiation set NS some deal to propose. It has no negotiation history to guide its choice. Similarly, when an agent is forced to 'backtrack' during negotiation, because it cannot make a concession, it must suggest a deal that is not simply a modification of the proposals it has already made. In order to make such a selection, we shall assume a set $T_i \subseteq NS$ of deals that are 'on the table': deals that have previously been proposed by agent i. At the start of negotiation, this set will be empty. The intuition is that, when conceding or backtracking, an agent is not allowed to propose a deal that it has previously put forward. Let bp'_i be the set of deals in NS which give agent i the equal best payoff from all the deals in the negotiation set that are not 'on the table', i.e., that have not previously been proposed by i.

$$bp'_i = \{\sigma \mid \sigma \in NS - T_i \text{ and } c_i(\sigma) = \min\{c_i(\sigma') \mid \sigma' \in NS - T_i\}\}.$$

In addition, it seems reasonable to require that i only proposes deals in bp_i that minimize factory costs. The set bp_i contains only deals that have this additional property.

$$bp_i = \{\sigma \mid \sigma \in bp'_i \text{ and } c_f(\sigma) = \min\{c_f(\sigma') \mid \sigma' \in bp'_i\}\}$$

An agent i's first proposal, and any 'backtrack' proposal that it makes will be required to be members of the set bp_i (with respect to sets T_i and NS). The set bp_i will represent the *best proposals that agent i is still able to make.*.

Who Should Concede?

The solution we propose for this problem is a generalized version of the so-called *Zeuthen strategy*, originally proposed in [14], and described in [8, pp43–49]. Intuitively, the problem faced by every agent at every round is: should I concede or not? How is an agent to make this decision? Zeuthen's idea was that the agent with the *least to lose* from concession should be the one to concede. To put it another way, the agent that should concede is the one for whom conceding represents the *least risk*. Zeuthen suggested that agent i could quantify its *willingness to risk conflict* at round t of negotiation in the following way [8, p43]:

$$risk_i^t = \frac{\text{utility agent } i \text{ loses by conceding}}{\text{utility agent } i \text{ loses by not conceding}}.$$

The utility an agent loses by not conceding is defined to be the utility of that agent's currently proposed deal. (Intuitively, if an agent does not concede, it may cause conflict, and lose the whole of its utility.) To determine the utility an agent loses by conceding, we let the agent *assume the worst*. This is a standard idea in game theory, and is embodied in such fundamental concepts as the minimax principle [3]. From the point of view of an agent i, the worst that can happen if it concedes is that it will end up having to carry out the currently proposed deal that is worst from its point of view. Writing σ_i^t for the deal proposed by agent i at round t in the protocol, we thus define the willingness of agent i to 'risk conflict' at round t, (denoted $risk_i^t$), as:

$$risk_i^t = \begin{cases} 1 & \text{if } utility_i(\sigma_i^t) = 0 \\ \frac{utility_i(\sigma_i^t) - \min\{utility_i(\sigma_j^t) \mid j \in Ag\}}{utility_i(\sigma_i^t)} & \text{otherwise.} \end{cases}$$

How Much Should be Conceded?

Suppose T_i is the set of deals that have been proposed by agent i through rounds 1 to t, and that at round t, agent i proposed deal σ_i^t. Agent i then discovers that it must concede. What should i's next proposal, σ_i^{t+1}, be? Ideally, i would find a deal in NS that represents a true, sensible concession. We suggest that such a concession should enjoy the following properties:

1. $\sigma_i^{t+1} \notin T_i$

   Proposing a deal that you proposed earlier is pointless.

2. $\sigma_i^{t+1} \succ_{Ag-\{i\}} \sigma_i^t$

   This condition states that i really *is* making a concession: the deal it will propose is at least as good for every other agent as its current offer, and actually better for at least one other agent. A subtlety of this condition is that a conceding agent cannot propose a concession deal that improves the lot of the rest of the group by 'rearranging' their utilities: it has to make at least one agent better off, and every other agent no worse off. The rationale behind this condition is that if an agent *did* rearrange the utilities, then it would make some agent worse off than its previously proposed deal, which would rule out that agent accepting the new offer.

3. $c_f(\sigma_i^{t+1}) < c_f(\sigma_i^t)$

   This condition requires that the deal improves the lot of the whole factory: there is no point in proposing a deal that makes some agent better off at the expense of the whole factory. (Note that condition (3) is *not* implied by condition (2); nor is (2) implied by (3).)

4. under the assumption that at round $t+1$, agent i proposes σ_i^{t+1}, and that $\forall k \in Ag$, $k \neq i$ implies that k proposes σ_k^t at round $t+1$, then $\exists j \in Ag$ for whom $risk_j^{t+1} < risk_i^{t+1}$

   The third condition requires that the concession made by the agent will be sufficient to change the *balance of risk* within the group: under the assumption that every other agent proposes the same deal, then on the next round, somebody else will have to concede. If we did not make this a requirement, then one agent might be forced to concede round after round until finally, it has conceded enough to shift the balance of risk. This would conflict with the simplicity criterion, and so this requirement states that i must concede *enough*, so that on the next round it need not concede.

5. $c_i(\sigma_i^{t+1}) = \min\{c_i(\sigma) \mid \sigma \in NS$ and σ satisfies conditions (1)–(4)$\}$.

   This final condition states that the deal i will offer is the minimal concession that i could make: the new offer is the least cost deal that i could propose which satisfied conditions (1)–(4).

These four properties represent *heuristics*, which (hopefully) guide the search to a solution. It may be that a conceding agent i is unable to find a deal that satisfies these conditions. In this case, that agent must 'backtrack', by recomputing bp_i. If bp_i is non-empty, then the agent proposes any member of this set; if bp_i *is* empty, then the agent withdraws from negotiation.

It is worth noting that, in the two agent case, conditions (1)–(3) are not required [8, p44]. It is the potential presence of more agents that introduces the need to focus an agent's proposal-making process through heuristics.

The Negotiation Algorithm

The algorithm will use the following variables:

- $t \in I\!N$ is a *round counter*;
- $T_i \subseteq NS$ represents the set of deals proposed by agent i thus far, for all $i \in Ag$;
- $Active \subseteq Ag$ represents the set of agents still active in negotiation, i.e., the set of agents that have not yet exhausted the set of deals they can possibly propose.

The negotiation algorithm is then as follows:

1. Set t to 1.
2. For each agent $i \in Ag$, set T_i to $\emptyset$.
3. Set $Active$ to Ag.
4. For each agent $i \in Ag$, compute bp_i.

   (Note that bp_i is guaranteed to be non-empty at this stage.)
5. For each agent $i \in Ag$, non-deterministically select a deal σ_i^1 from bp_i.
6. For each agent $i \in Active$, set T_i to $T_i \cup \{\sigma_i^t\}$.
7. Check for agreement. This is done by first computing the *agreement set*:

   $$\{\sigma \mid \sigma \in \bigcup_{i \in Ag} T_i \text{ and } \forall j \in Ag, \exists \sigma' \in T_j \text{ such that } utility_j(\sigma) \geq utility_j(\sigma')\}$$

   That is, the agreement set contains all those deals that have been proposed, such that those deals make every agent j at least as well off as a deal previously proposed by j.

   If the agreement set is non-empty, then agreement has been reached, and negotiation ends with the agents accepting any member of the agreement set.

8. For each agent $i \in \mathit{Active}$, compute risk_i^t.

9. Let g be the set of agents such that $\forall i \in g$, $\mathit{risk}_i^t = \min\{\mathit{risk}_j^t \mid j \in \mathit{Active}\}$, i.e., the agents with the (equal) least willingness to 'risk conflict'.

10. For each agent $i \in g$:

    - if i can make a concession deal, that satisfies the properties listed earlier, then set σ_i^{t+1} — the deal i will propose on the next round — to be such a deal;
    - if i cannot make a concession deal, then compute bp_i:
      - if $bp_i \neq \emptyset$ (i.e., the agent has not exhausted the deals it could possibly propose), then set σ_i^{t+1} to be any member of bp_i;
      - if $bp_i = \emptyset$, then set Active to $\mathit{Active} - \{i\}$ (i.e., i withdraws from negotiation).

11. For each agent $i \in \mathit{Active} - g$, set σ_i^{t+1} to be σ_i^t, i.e., every other agent's offer will remain unchanged.

12. Set t to $t + 1$.

13. Goto (6).

Steps (1) to (5) represent initialization for the first round of negotiation. Steps (8) and (9) represent the process of deciding who will concede; step (10) defines what such conceding agents will do. Using this algorithm, agents will systematically search the negotiation set, looking for an agreement deal. It is not difficult to see that the algorithm therefore has the following important property:

Theorem 3 *After a finite number of steps, the agreement set will be non-empty, and thus the negotiation algorithm is guaranteed to terminate with agreement.*

PROOF: Assume agreement is never reached. Steps (8) to (10) ensure that, on every round, some non-empty subset of active agents will either withdraw or else propose deals that they have not previously suggested. An agent i will only withdraw if $T_i = NS$, i.e., if i has proposed all possible deals. Eventually, therefore, every agent would propose all possible deals. But in this case, where $T_i = NS$ for all $i \in Ag$, the agreement set would be non-empty, and so agreement would have been reached. Thus the assumption is false. □

3.3 That Example Again

Let us now dry run the negotiation algorithm using the example presented in sections 2.4 and 3.3. We begin by setting t to 1 and T_i to $\emptyset$, for all $i \in Ag$. The variable Active is set to Ag. Every agent must then choose a first deal, which in turn requires computing bp_i for $i \in \{1, 2, 3\}$ (recall that the negotiation set, NS, is $\{\sigma_1, \sigma_2, \sigma_{12}\}$). We have:

$$bp_1 = \{\sigma_2\} \qquad bp_2 = \{\sigma_1, \sigma_{12}\} \qquad bp_3 = \{\sigma_1, \sigma_2, \sigma_{12}\}.$$

The algorithm is non-deterministic, in that every agent i is allowed to choose at random some member of bp_i. Suppose that agent 1 chooses σ_2 (it has no choice), agent 2 chooses σ_1, and agent 3 chooses σ_{12}. There is no agreement, and so we compute every agent's willingness to risk:

$$risk_1^1 = \frac{1}{3} \qquad risk_2^1 = 1 \qquad risk_3^1 = 0.$$

Agent 3 must concede. Agents 1 and 2 put forward their previous proposals on round 2, and agent 3 attempts to compute a concession deal. There are only two possibilities in the negotiation set: σ_1 and σ_2. However, neither of these represents a 'true' concession, so agent 3 must recompute bp_3. Clearly, bp_3 will now contain σ_1 and σ_2, so agent 3 non-deterministically selects one of these for round 2. Suppose it selects σ_1. We update the round counter to 2, and move to the next round.

On this round, agent 1 proposes σ_2, while agents 2 and 3 propose σ_1. Agreement has not yet been reached, and so we compute $risk_i^2$, for $i \in \{1, 2, 3\}$: we find that the risk values remain unchanged. Agent 3 must concede once more. The agent again tries to find a concession deal, but fails. It therefore recomputes bp_3, which now contains just σ_2, which becomes agent 3's next proposal. The round counter t is updated to 3.

In round 3, agents 1 and 3 propose σ_2, and agent 2 proposes σ_1. We find that the risk values remain unchanged once more, and so agent 3 is required to concede yet again. As agent 3 cannot make a 'true' concession, it recomputes bp_3, but this time finds it is empty: it has exhausted the set of all deals it could propose. Agent 3 therefore drops out of negotiation, and $Active$ is set to $\{1, 2\}$. The round counter is updated to 4.

On round 4, agents 1 and 2 put forward the same deals they proposed on the previous round: σ_2 and σ_1 respectively. On computing risk, we find:

$$risk_1^4 = \frac{1}{3} \qquad risk_2^4 = 1.$$

Agent 1 must concede. It cannot make a 'true' concession, since the deal it currently offers is globally optimal. However, recomputing bp_1 gives two possibilities: σ_1 and σ_{12}. Whichever of these proposals agent 1 makes, agreement will be reached on round 5, as agents 2 and 3 have both previously proposed deals that make them no better off. Note that all of the three original possibilities (σ_1, σ_2, and σ_{12}) were globally optimal, and so negotiation was bound to conclude with a globally optimal solution. However, σ_1 and σ_{12} (the two possible conclusions to the algorithm) represent *fair* cost distributions, when compared to σ_2.

4 From Theory to Practice

As we noted earlier, this paper is not simply an abstract study. The PSP is a real-world problem, for which we intend to implement a negotiation-based solution. Our aim in this section is therefore to consider how well the theory of negotiation — as represented in the algorithm above — matches up to the computational reality. In particular, we discuss what simplifications and assumptions must be made in order to make the algorithm amenable to implementation.

To see the practical problems associated with the algorithm, consider an agent computing the utility of a deal with respect some encounter. This computation requires that the agent is able to determine both the cost of the deal, and the worst that it could do with respect to the encounter. Computing the cost of a deal is done via an agent's cost function, c_i. It is unlikely that a real factory could find a simple mathematical function that gives the cost of a deal to a factory cell. The computation of cost is therefore likely to be done using heuristics determined through experience. Such heuristics obviously introduce the possibility of error into the computation of cost. In some factories, with particularly complex products and production processes, it may be that no cost information can be reliably obtained: in this case, the computation of cost becomes an educated guess. With respect to finding the worst an agent could do in an encounter, we can see similar problems. An agent cannot enumerate all possible production sequences in order to find the worst; even if it could enumerate them, the cost information is obtained by heuristic methods. So the worst an agent could do in an encounter would also have to be computed heuristically.

The computation of utility is the foundation upon which the algorithm is constructed: an agent's decision making rests entirely upon the determination of this value. Clearly, if there is any element of uncertainty associated with the value, then the algorithm loses its desirable — provable — properties. An obvious question is then how well the algorithm performs when such uncertainty is present.

To summarize, there are two key problems associated with the algorithm from a practical point of view:

- it assumes that we can allocate each agent (production cell) an accurate cost function, whereas in practice, costs would be estimated heuristically;
- it assumes that agents have unlimited reasoning (computational) ability, in that, for example, they can search through the set of all production sequences in order to find the one that maximizes their cost, or represents the minimum concession deal, etc.

5 Concluding Remarks

We have presented the production sequencing problem: a common, important problem that occurs in factories throughout the world, and given a formal definition this problem. We proposed solving this problem using negotiation, and to this end, we recast the problem domain in negotiation theoretic terms. We then presented a negotiation algorithm for the problem, based on the Monotonic Concession Protocol with Zeuthen strategy, as described in [8]. However, we found that many of the assumptions that underpin this theoretically attractive strategy make it unworkable in practice.

With respect to future work, there are many obvious issues to be addressed. For example, as we observed in the preceding section, it is not clear how the algorithm will perform in the presence of uncertainty: experimental investigation is required in order to settle the question. Also, we need more experience with domain-specific negotiation heuristics for the MPSD. Again, experimental investigation would be appropriate in order to determine the performance of such heuristics.

Acknowledgments: This work was carried out while the first author was a visiting researcher at Daimler-Benz research institute in Berlin. Thanks to Kurt Sundermeyer, who made the visit possible, and everyone at DB for being so friendly.

References

[1] A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[2] N. R. Jennings, L. Z. Varga, R. P. Aarnts, J. Fuchs, and P. Skarek. Transforming standalone expert systems into a community of cooperating agents. *International Journal of Engineering Applications of Artificial Intelligence*, 6(4):317–331, 1993.

[3] R. D. Luce and H. Raiffa. *Games and Decisions*. John Wiley & Sons, 1957.

[4] K. Mori, H. Torikoshi, K. Nakai, and T. Masuda. Computer control system for iron and steel plants. *Hitachi Review,* 37(4):251–258, 1988.

[5] R. E. Morley and C. Schelberg. An analysis of a plant-specific dynamic scheduler. In *Proceedings of the NSF Workshop on Dynamic Scheduling*, Cocoa Beach, Florida, 1993.

[6] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley: Reading, MA, 1994.

[7] H. V. D. Parunak. Applications of distributed artificial intelligence in industry. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed AI*. John Wiley & Sons, 1995. (To appear).

[8] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. The MIT Press: Cambridge, MA, 1994.

[9] S. J. Russell and E. Wefald. *Do the Right Thing — Studies in Limited Rationality*. The MIT Press: Cambridge, MA, 1991.

[10] U. M. Schwuttke and A. G. Quan. Enhancing performance of cooperating agents in real-time diagnostic systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93),* pages 332–337, Chambéry, France, 1993.

[11] R. Steeb, S. Cammarata, F. A. Hayes-Roth, P. W. Thorndyke, and R. B. Wesson. Distributed intelligence for air fleet control. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 90–101. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[12] R. Weihmayer and H. Velthuijsen. Application of distributed AI and cooperative problem solving to telecommunications. In J. Liebowitz and D. Prereau, editors, *AI Approaches to Telecommunications and Network Management*. IOS Press, 1994.

[13] T. Wittig, editor. *ARCHON: An Architecture for Multi-Agent Systems*. Ellis Horwood: Chichester, England, 1992.

[14] F. Zeuthen. *Problems of Monopoly and Economic Warfare*. G. Routledge and Sons: London, 1930.