Multi-agent based beam search for intelligent production planning and scheduling

by

S.G. Kang and S. H. Choi*
Department of Industrial and Manufacturing Systems Engineering
The University of Hong Kong,
Pokfulam Road,
Hong Kong.

* Corresponding author, email: shchoi@hku.hk

Multi-agent based beam search for intelligent production planning and scheduling

Abstract

Production planning and scheduling is a long standing research area of great practical value, while industrial demand for production planning and scheduling systems is acute. Regretfully, most research results are seldom applied in industry because existing planning and scheduling methods can barely meet the requirements for practical applications. This paper identifies four major requirements, namely generality, solution quality, computation efficiency, and implementation difficulty, for practical production planning and scheduling methods. Based on these requirements, method, multi-agent based beam search (MABBS), is developed. It seamlessly integrates the multi-agent system (MAS) method and beam search (BS) method into a generic multi-stage multi-level decision making (MSMLDM) model to systematically address all the four requirements within a unified framework. A script language, called EXASL, and an open software platform are developed to simplify the implementation of the MABBS method. For solving complex real-world problems, an MABBS-based prototype production planning, scheduling and execution system is developed. The feasibility and effectiveness of this study is demonstrated with the prototype system and computation experiments.

Keywords: production planning, scheduling, multi-agent systems, beam search, decision making, knowledge representation.

1. Introduction

Production planning and scheduling are long-standing research problems, which are critical to the efficiency of manufacturing systems (Pinedo 2002). Planning determines which activities to perform, while scheduling allocates resources over time to these activities, so that production tasks can be accomplished timely and cost-effectively (Smith 2003). Despite the tremendous effort made on production planning and scheduling, their application to real-world problems remains very limited (Jacobs and Weston 2007). Therefore, further effort is needed to narrow the gap between research and application in this area. The major research challenges include the following aspects.

- The NP-hard property of planning and scheduling
- Complex constraints, preferences and objectives
- Changes
- Integration of planning and scheduling

Firstly, planning and scheduling generally belong to a class of NP-hard problems, to which the optimal solutions usually cannot be found efficiently (Jain and Meeran 1999). They are often studied as typical combinatorial optimization problems. Various exact or approximate optimization algorithms have been developed to calculate optimal or near optimal solutions, such as branch and bound (Lageweg and Rinnooy 1977), beam search (Lowerre 1976) (Ow and Morton 1988) (Sabuncuoglu and Bayiz 1999), tabu search (Amico and Trubian 1993), simulated annealing (Van Laarhoven et al. 1992), genetic algorithms (Kobayashi 1995) (Jensen 2003), shift bottleneck procedure (Adams et al. 1988), machine order space search (Yang 2003), etc. In order to measure and compare the performance of different algorithms, some benchmark problems have been developed (Jain and Meeran 1999).

Secondly, manufacturing systems are complex. As a result, real-world production planning and scheduling problems often involve many constraints, preferences, and multiple conflicting optimization objectives (Smith 2003). However, most optimization algorithms tend to focus on tackling the NP-hard property with idealized problems, which often overlook the complex practical factors. Therefore, these algorithms usually cannot be applied directly to real-world problems, which are difficult to model in a rigorous mathematical approach. In this connection, other approaches have been developed, such as simulation-based approach (Marcus 1990) (Harmonosky 1995) (Musselman et al. 2002), expert system based approach (Bruno et al. 1986; Bruno and Morisio 1987), multi-agent based approach (Shen 2002; Shen et al. 2006) (Caridi and Cavalieri 2004), etc. These approaches consider some practical factors, but they often cannot effectively address the NP-hard property and cannot provide high quality solutions (Smith 1992) (Shen 2002) (Caridi and Cavalieri 2004). Indeed, to solve practical production planning and scheduling problems, it is necessary to consider their combinatorial, explosive nature (NP-hard property) and the complex practical factors (constraints, preferences, and objectives) simultaneously.

Thirdly, most manufacturing systems are dynamic, in which planning and scheduling are ongoing processes of responding to unexpected and evolving circumstances (Smith 2003). In daily operation of manufacturing systems, there are indeed no pure planning and scheduling problems, but re-planning and rescheduling ones. How to response promptly to changes and avoid “nervousness” caused by frequent and drastic schedule changes is critical to practical production planning and scheduling. This remains a research challenge.

Finally, although planning and scheduling are traditionally studied separately, they are indeed not clearly separable in practice. The classical “waterfall” approach of planning and scheduling leads to lengthy and inefficient problem-solving cycles (Smith 2003). It is desirable to integrate planning and scheduling into a unified problem-solving framework, as pointed out by Myers (Myers and Smith 1999) and Tan (Tan and Khoshnevis 2000).

Take in

Figure 1: Traditional research methodology

From the discussion above, it is apparent that a lot of research has been done to tackle the challenges in production planning and scheduling. However, industrial application of planning and scheduling research remains very limited, and most results cannot be applied directly to real-world problems. Such a situation seems to be the consequence of the traditional research methodology, as illustrated in figure 1. Real-world problems are usually complex, but most research tends to focus on certain aspects of challenges and ignore the others. Although this methodology may address some specific challenges, it cannot effectively lead to practical solutions. This explains why research results usually alienate the industrial practices of planning and scheduling (Jacobs and Weston 2007).

In order to develop methods and tools that can be directly applicable to complex real-world problems, we should improve the methodology to systematically study the challenges within a unified framework. Accordingly, the philosophies and techniques in different research approaches should be effectively integrated into such a common framework.

Based on the above observation, a knowledge-directed opportunistic search method, called the multi-agent based beam search (MABBS), is developed to provide a generic approach that can seamlessly incorporate knowledge and search together, and can trade off between solution quality and computation efficiency. Thus, it provides a promising approach to complex real-world production planning and scheduling.

This paper is organized as follows. The next section briefly reviews multi-agent based planning and scheduling, which is closely related to this study. Section 3 analyzes the production planning and scheduling problems, and identifies four major requirements for practical production planning and scheduling methods. Section 4 describes the MABBS method in detail. Section 5 describes the software implementation of the MABBS method. Section 6 discusses the MABBS-based prototype production planning and scheduling system and the computation experiments. Section 7 concludes this paper and discusses further research directions.

2. Literature Review

This section briefly reviews MAS-based production planning and scheduling, and identifies research opportunities for this study.

Over the past two decades, there has been a substantial amount of research work on adoption of multi-agent system (MAS) (Caridi and Cavalieri 2004). A multi-agent system contains multiple intelligent agents which communicate and cooperate with each other to solve complex problems in a distributed manner. It is a natural abstraction of a complex manufacturing system and has been investigated in recent years as a promising approach to production planning, scheduling and control.

Communication and collaboration among intelligent agents is a central problem in MAS. Accordingly, MAS communication and collaboration methods, standards, and software platforms, such as contract net protocol (Smith 1980), FIPA specifications (O'Brien and Nicol 1998), and JADE (Bellifemine et al. 2001), have been developed to facilitate construction of multi-agent systems. Based on these techniques, some prototype MAS-based production planning and scheduling systems have been developed for FMS (Kouiss et al. 1997) (Wang et al. 2008), wafer fabs (Mönch et al. 2006), and supply chain coordination (Fox et al. 2000) (Dangelmaier et al. 2005), etc. Detailed reviews of multi-agent system based production planning and scheduling can be found in (Caridi and Cavalieri 2004) and (Shen 2002; Shen et al. 2006).

Despite the promising prospect of MAS-based production planning and scheduling, more research effort is needed to make it practicable (Mönch and Stehli 2006). Firstly, the MAS-based production scheduling systems are developed based on the belief that multiple intelligent agents can communicate and collaborate toward global optimal solutions. However, how this can be achieved remains obscure (Shen 2002) (Caridi and Cavalieri 2004). Secondly, the computation efficiency of MAS-based systems is often unpredictable, and the communication overload becomes very heavy when the number of agents is large (Caridi and Cavalieri 2004). Thirdly, it is indeed a non-trivial task to implement an MAS-based production planning and scheduling system, even with the existing MAS tools and techniques (Mönch and Stehli 2006).

Recently, some researchers have attempted to tackle these issues. For example, Shen (2002) pointed out that effective integration of the traditional centralized approach with MAS is a promising research area of great potential. Based on this notion, Wang developed a prototype MAS system and combined the MAS method with the beam search method to improve the solution quality (Wang et al. 2008). Mönch pointed out that the generic MAS techniques usually cannot provide enough support for developing complicated production planning and scheduling systems, and proposed a framework, called ManufAg (Mönch and Stehli 2006), to go beyond the generic MAS techniques.

Based on the observation above, this study attempts to develop an effective method to take advantage of the strength of MAS and to improve its solution quality and computation efficiency through a “system level” beam search. The proposed method is subsequently implemented into a generic and reusable software platform to facilitate development of complex practical production planning and scheduling systems.

3. Problem analysis

Production planning and scheduling are dynamic, complex decision-making processes, which usually lead to numerous possible solutions. The purpose of a production planning and scheduling method is to find a feasible and high quality solution in the huge solution space effectively.

In order to analyze the production planning and scheduling problems, a set of formulations are introduced in this paper. These formulations are not strictly in mathematical equations, but in a similar form which can help explain some abstract ideas. Firstly, let's formulate a production planning and/or scheduling method as follows.

$$W_M = M(W_0, C) \tag{1}$$

In formula 1, *M* denotes a planning and/or scheduling method. W_0 denotes the input of *M*, W_M denotes the output of *M*, and *C* denotes the constraints. The meaning of formula (1) is that given the input W_0 and constraints *C*, a planning and/or scheduling method *M* generates the output W_M. This can be illustrated in figure 2.

Take in

Figure 2: Illustration of a planning and/or scheduling method

Formula 1 can be rewritten in another form, as shown in formula 1'. It means W_M is the result of W_0 going through a process prescribed by M and C.

$$W_M = W_0 + \langle M, C \rangle \tag{1'}$$

W_0 includes two parts, P_0 and S_0. P_0 denotes the problem, and S_0 the initial solution. This can be represented by formula (2). Similarly, W_M can be represented by formula (3).

$$W_0 = P_0 + S_0 \tag{2}$$

$$W_M = P_M + S_M \tag{3}$$

For W_M, it should satisfy the following conditions:

$$|P_M| = 0 \tag{4}$$

$$\forall c \in C \quad c(S_M) = 0 \tag{5}$$

Formula 4 means that all the problems from the input have been solved, while formula 5 implies that the final solution S_M satisfies all the given constraints. In formula 5, $c(S_M) = 0$ means constraint c is satisfied, while $c(S_M) = 1$ means constraint c is violated.

In this paper, production planning refers to production process planning, and scheduling refers to the low-level shop floor control functions. They can be defined with the generic formulation above, as shown in table 1.

Take in

Table 1: Definition of planning and scheduling problems with the generic framework

Let Λ denotes all the possible solutions, i.e. $\Lambda = \{S_M\}$. Due to the combinatorial, explosive nature of production planning and scheduling problems, $|\Lambda|$ grows exponentially with the problem size, and it can be extremely large. It is indeed a big challenge to find the optimal solution from the huge solution space Λ. This can be formulated as follows.

$$\textit{Find } S^* \in \Lambda \quad o(S^*) = \min o(S_M) \qquad (6)$$

In formula (6), $S*$ denotes the optimal solution, and o denotes some performance measures.

Traditionally, scheduling is studied as an optimization problem, and a lot of optimization methods were developed. Various optimization algorithms compete with each other on solution quality and computation speed. Due to the NP hard property of the production planning and scheduling problems, it is usually very difficult to find the optimal solution and to prove a solution is optimal. For example, it took 25 years to find the optimal solution for the famous 10×10 job shop scheduling problem in the scheduling research history (Jain and Meeran 1999). When the problem size becomes larger, no method can effectively get the optimal solution. Indeed, some large benchmark problems created many years ago have not yet been solved optimally.

Regretfully, optimization techniques are seldom adopted in industry to solve the real-world problems. According to Jacobs (2007), little has changed since the late 1970s in the underling production planning and scheduling logic in ERP systems. Most systems are now just executing the old logic with much faster hardware and in real-time, although the current ERP technology can provide an information rich environment that is ripe for very intelligent planning and scheduling logic. Herrmann

(2005) suggested researchers to focus less on the mathematical properties of the combinatorial optimization problems and more on the complex production scheduling systems that still need improvement. Based on this observation, we may need to think about the following two problems:

- Why are the scheduling research results difficult to apply?
- How to design practical production planning and scheduling methods?

From this perspective, four major requirements are identified for practical production planning and scheduling methods.

- Generality
- Solution quality
- Computation efficiency
- Implementation difficulty

Generality means that a practical production planning or scheduling method can be applied to complex real-world problems, instead of being limited to solving some specific idealized problems. In the scheduling literatures, scheduling problems are usually classified into various categories, such as job shop scheduling problems, flow shop scheduling problems, etc. Each category can be further categorized according to some criteria. Different types of problems have different problem structure, solution structure, constraints, and optimization objectives. According to formulas (1) to (5), different types of scheduling problems may vary on P_0, S_0, C, S_M, and o. Most scheduling research is based on certain types of scheduling problems. However, in industrial practice, the complex manufacturing systems call for more complicate problem structure, solution structure, constraints and optimization objectives than the theoretical models. Generality is indeed a major obstacle that hinders applications of optimization methods in industry.

Solution quality means the solution should be good enough to improve the efficiency of the manufacturing systems. Although it is generally not realistic to pursue optimal solutions for practical purposes, high solution quality remains a major goal for a production planning and scheduling method.

Computation efficiency means how much computation time is required to get the result. As manufacturing systems are dynamic systems that are subject to frequent changes, the planning and scheduling functions should be able to produce timely responses to accommodate the changes. This requires the production planning and scheduling method to produce results quickly.

Implementation difficulty means how much effort is required to develop a practical system based on the production planning and scheduling method. This depends largely on how to develop reusable software packages which can be conveniently applied to complex practical problems.

If a production planning and scheduling method can meet all the above requirements, it is generic enough to model the complex real-world production planning and scheduling problems and can accommodate the complex constraints. Moreover, it can produce high quality solutions quickly and is convenient and cost-effective to develop practical production planning and scheduling systems with reusable software packages or tools. Hence, it would be a practical production planning and scheduling method.

However, the existing planning and scheduling methods do not meet all these requirements. Most scheduling optimization algorithms pursue solution quality and computation efficiency, but they are usually not generic enough to handle the complex real-world problems. The simulation-based scheduling methods and multi-agent scheduling methods are more powerful on modelling realistic manufacturing systems, but they cannot produce high quality solutions. The expert system based scheduling methods also suffer from poor solution quality. This explains why most existing scheduling methods are not used to solve real-world problems.

With the understanding above, this paper attempts to systematically consider all the four requirements into a unified framework for developing a practical production planning and scheduling method that can be directly applied to industrial practice.

4. The Multi-Agent Based Beam Search Method

The design of the multi-agent based beam search (MABBS) method is based on the four main requirements discussed in the previous section. For the first requirement, a *multi-stage multi-level decision making* (MSMLDM) model is developed to provide a generic framework to model complex decision making processes by generalization of production planning and scheduling problems. For the second and third requirements, a knowledge-directed opportunistic search (KDOS) method is developed to achieve a balance between solution quality and computation efficiency. For the fourth requirement, an open software platform and a script language, called *embedded extensible application script language* (EXASL), are developed to provide a flexible and reusable computer software tool to facilitate quick and cost-effective application of the MABBS method to various complex real-world problems. Solutions for the first three requirements are discussed in this section, while the solution for the fourth one will be discussed in the next section.

This section is organized as follows. Section 4.1 describes the MSMLDM model; section 4.2 describes the knowledge directed opportunistic search method; and section 4.3 describes the MABBS method.

4.1 The multi-stage multi-level decision making model

Production planning and scheduling problems are complex decision making problems (Pinedo 2002) (Herrmann 2004). Therefore, for each given problem, there is a set of decision variables $D=\{d_1, d_2, \ldots, d_n\}$. If the values of the decision variables are determined, then the final solution is determined. We can formulate this as follows.

$$W_M = W_0 + \langle F, D_M \rangle \qquad (7)$$

In formula 7, W_0 means the input, W_M the output, and D_M a set of values for the decision variables, while F is a transformation function, which can deterministically transform W_0 into W_M according to D_M.

From formulas 1' and 7, we can induce formula 8:

$$\because\ W_M = W_0 + \langle M, C \rangle = W_0 + \langle F, D_M \rangle$$

$$\therefore\ \langle M, C \rangle = \langle F, D_M \rangle \qquad (8)$$

Formula 8 indicates that the function of a production planning and scheduling method is equivalent to the following two steps:

- Determine the values of the decision variables D_M;
- Perform a deterministic transformation from W_0 to W_M according to D_M.

A problem may be solved with various methods. For example, for a given problem, W_0 and C, there are two different methods, M_1 and M_2. According to formula 8,

$$\langle M_1, C \rangle = \langle F, D_{M1} \rangle \qquad (9)$$

$$\langle M_2, C \rangle = \langle F, D_{M2} \rangle \qquad (10)$$

By comparing formulas 9 and 10, we can see for a given problem (W_0 and C) that the only difference between two solution methods (M_1 and M_2) is how they determine the values of the decision variables. As each decision variable can be assigned with a set of alternative values (for discrete decisions), the combination of the decision variables results in enormous possibilities. Let $\Gamma = \{D_M\}$. Then, the essential problem of a production planning and scheduling method is to:

$$\textit{Find} \quad D^* \in \Gamma, \ \textit{satisfies}\,(5) \ \textit{and} \ \textit{optimizes}\,(6) \qquad (11)$$

Different methods use different strategies to determine D_M. This results in different solution qualities, computation efficiencies, as well as different degrees of generality. Some methods, such as the branch and bound method, the genetic algorithms, the tabu-search method, and the simulated annealing method, search Γ extensively to find a high quality D_M. These methods usually can find high quality solutions, but they are time-consuming and difficult to model complex manufacturing systems. Some other methods, including the expert based scheduling and multi-agent based scheduling, rely on certain kinds of knowledge. Such methods can accommodate some kinds of human intelligence but cannot guarantee solution quality. There are also methods that use some myopic and greedy policies. These include dispatching rules and simulation based scheduling, which tend to be simple and applicable to complex real-world situations but suffer from poor solution quality.

Based on the observation above, we may need to consider the following problem: *How to design a method that can produce high quality solutions, accommodate human intelligence, and be generic enough to model complex real-world manufacturing systems*?

From this perspective, a generic multi-stage multi-level decision making (MSMLDM) model is developed to provide a framework for complex decision making processes. MSMLDM model is based on two types of decompositions:

- Parallel decomposition
- Sequential decomposition

Parallel decomposition means to decompose a complex decision making method into several alternative methods, each of which leads to a different solution. For example, a decision making method M is decomposed into two alternative methods M_1 and M_2. M_1 leads to solution W_{M1} and M_2 to solution W_{M2}, as illustrated in figure 3.

Take in

Figure 3: Parallel decomposition

This can be represented as follows.

$$W_M = W_0 + \langle M, C\rangle = W_0 + \begin{cases} \langle M_1, C\rangle \\ \langle M_2, C\rangle \end{cases} \tag{12}$$

From formula 12, we can infer formula 13:

$$\langle M,C\rangle = \begin{cases}\langle M_1,C\rangle \\ \langle M_2,C\rangle\end{cases} \tag{13}$$

By combining formula 8 and formula 13, we can get formula 14:

$$\langle F,D_M\rangle = \langle M,C\rangle = \begin{cases}\langle M_1,C\rangle \\ \langle M_2,C\rangle\end{cases} = \begin{cases}\langle F,D_{M1}\rangle \\ \langle F,D_{M2}\rangle\end{cases} \tag{14}$$

From formula 14, we can get formula 15:

$$D_M = \begin{cases} D_{M1} & (M_1 \text{ is used}) \\ D_{M2} & (M_2 \text{ is used})\end{cases} \tag{15}$$

Formula 15 indicates that the values of the decision variables D_M in the final solution depend on which method is used. In other words, by choosing alternative methods, we can control the values of the decision variables. In general, we can represent the parallel decomposition as follows.

$$\langle M,C\rangle = \overset{n}{\underset{i=1}{\{}} \langle M_i,C\rangle \quad (n \geq 2) \tag{16}$$

Take in

Figure 4: Sequential decomposition

Sequential decomposition means to decompose a decision making problem into a sequence of steps. For example, a decision making method M is decomposed into two steps M_1 and M_2. M_1 leads to solution W_{M1}, which is used by M_2 for the subsequent the decision making process to produce the final solution W_M, as illustrated in figure 4.

This can be formulated as follows.

$$W_M = W_0 + \langle M,C\rangle = W_1 + \langle M_2,C_2\rangle = W_0 + \langle M_1,C_1\rangle + \langle M_2,C_2\rangle \tag{17}$$

From formula 17 we can infer formula 18:

$$\langle M,C\rangle = \langle M_1,C_1\rangle + \langle M_2,C_2\rangle \tag{18}$$

By combining formulas 8 and 18, we get formula 19:

$$\langle F,D_M\rangle = \langle M,C\rangle = \langle M_1,C_1\rangle + \langle M_2,C_2\rangle = \langle F,D_{M1}\rangle + \langle F,D_{M2}\rangle = \langle F,D_{M1}+D_{M2}\rangle \tag{19}$$

From formula 19, we can get formula 20:

$$D_M = D_{M1} + D_{M2} \tag{20}$$

Formula 20 indicates that method M_1 determines the values of part of the decision variables (assigned in D_{M1}), while method M_2 determines the values of the other decision variables (assigned in D_{M2}). They work together to determine all the values of the decision variables (D_M).

In general, sequential decomposition can be represented by formula 21. It can indeed transform a complex decision making problem into a sequence of simpler decision making problems, i.e., a multi-stage decision making process.

$$\langle M,C\rangle = \sum_{i=1}^{n}\langle M_i,C_i\rangle \qquad (n\geq 2) \tag{21}$$

For each M_i in formulas 16 and 21, it can be further decomposed with formulas 16 and 21. In other words, parallel decomposition and sequential decomposition can be used recursively. With such a framework, a very complex decision making problem can be decomposed into a number of simple problems. This can be illustrated with the following example. Suppose a method M can be sequentially decomposed into three methods, M_1, M_2 and M_3. M_1 can be further parallel decomposed into two methods M_{11} and M_{12}. M_{11} can be further sequentially decomposed into M_{111} and M_{112}. M_{111} can be further parallel decomposed into M_{1111} and M_{1112}. M_2 can be further parallel decomposed into M_{21}, M_{22}, and M_{23}. M_3 can be further sequentially decomposed into M_{31}, M_{32}. M_{31} can be further parallel decomposed into M_{311}, M_{312}, M_{313} and M_{314}. This can be represented as follows.

$$\begin{aligned}
\langle M,C\rangle &= \langle M_1,C_1\rangle + \langle M_2,C_2\rangle + \langle M_3,C_3\rangle \\
&= \begin{cases}\langle M_{11},C_1\rangle \\ \langle M_{12},C_1\rangle\end{cases} + \begin{cases}\langle M_{21},C_2\rangle \\ \langle M_{22},C_2\rangle \\ \langle M_{23},C_2\rangle\end{cases} + \left(\langle M_{31},C_{31}\rangle + \langle M_{32},C_{32}\rangle\right) \\
&= \begin{cases}\langle M_{111},C_{111}\rangle + \langle M_{112},C_{112}\rangle \\ \langle M_{12},C_1\rangle\end{cases} + \begin{cases}\langle M_{21},C_2\rangle \\ \langle M_{22},C_2\rangle \\ \langle M_{23},C_2\rangle\end{cases} + \begin{cases}\langle M_{311},C_{31}\rangle \\ \langle M_{312},C_{31}\rangle \\ \langle M_{313},C_{31}\rangle \\ \langle M_{314},C_{31}\rangle\end{cases} + \langle M_{32},C_{32}\rangle \\
&= \begin{cases}\begin{cases}\langle M_{1111},C_{111}\rangle \\ \langle M_{1112},C_{111}\rangle\end{cases} + \langle M_{112},C_{112}\rangle \\ \langle M_{12},C_1\rangle\end{cases} + \begin{cases}\langle M_{21},C_2\rangle \\ \langle M_{22},C_2\rangle \\ \langle M_{23},C_2\rangle\end{cases} + \begin{cases}\langle M_{311},C_{31}\rangle \\ \langle M_{312},C_{31}\rangle \\ \langle M_{313},C_{31}\rangle \\ \langle M_{314},C_{31}\rangle\end{cases} + \langle M_{32},C_{32}\rangle
\end{aligned} \tag{22}$$

Parallel decompositions can lead to multiple solution paths. In the above example, there are $(2+1)\times 3\times 4 = 36$ different solution paths, each of which leads to a solution. The 36 solutions paths are listed below.

1. $\langle M_{1111},C_{111}\rangle + \langle M_{112},C_{112}\rangle + \langle M_{21},C_2\rangle + \langle M_{311},C_{31}\rangle + \langle M_{32},C_{32}\rangle = \langle F,D_{P1}\rangle$
2. $\langle M_{1111},C_{111}\rangle + \langle M_{112},C_{112}\rangle + \langle M_{21},C_2\rangle + \langle M_{312},C_{31}\rangle + \langle M_{32},C_{32}\rangle = \langle F,D_{P2}\rangle$
3. $\langle M_{1111},C_{111}\rangle + \langle M_{112},C_{112}\rangle + \langle M_{21},C_2\rangle + \langle M_{313},C_{31}\rangle + \langle M_{32},C_{32}\rangle = \langle F,D_{P3}\rangle$

4. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P4}\rangle$
5. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P5}\rangle$
6. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P6}\rangle$
7. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P7}\rangle$
8. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P8}\rangle$
9. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P9}\rangle$
10. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P10}\rangle$
11. $\langle M_{1111}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P11}\rangle$
12. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P12}\rangle$
13. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P13}\rangle$
14. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P14}\rangle$
15. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P15}\rangle$
16. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P16}\rangle$
17. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P17}\rangle$
18. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P18}\rangle$
19. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P19}\rangle$
20. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P20}\rangle$
21. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P21}\rangle$
22. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P22}\rangle$
23. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P23}\rangle$
24. $\langle M_{1112}, C_{111}\rangle + \langle M_{112}, C_{112}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P24}\rangle$
25. $\langle M_{12}, C_{1}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P25}\rangle$
26. $\langle M_{12}, C_{1}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P26}\rangle$
27. $\langle M_{12}, C_{1}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P27}\rangle$
28. $\langle M_{12}, C_{1}\rangle + \langle M_{21}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P28}\rangle$
29. $\langle M_{12}, C_{1}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P29}\rangle$
30. $\langle M_{12}, C_{1}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P30}\rangle$
31. $\langle M_{12}, C_{1}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P31}\rangle$
32. $\langle M_{12}, C_{1}\rangle + \langle M_{22}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P32}\rangle$
33. $\langle M_{12}, C_{1}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{311}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P33}\rangle$
34. $\langle M_{12}, C_{1}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{312}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P34}\rangle$
35. $\langle M_{12}, C_{1}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{313}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P35}\rangle$
36. $\langle M_{12}, C_{1}\rangle + \langle M_{23}, C_{2}\rangle + \langle M_{314}, C_{31}\rangle + \langle M_{32}, C_{32}\rangle = \langle F, D_{P36}\rangle$

Hence we can get the following formula.

$$D_M = \begin{cases} D_{p1} & (solution\ \ path\, 1\, is\ used) \\ D_{p2} & (solution\ \ path\ 2\ is\ used) \\ \cdots & \qquad\qquad \cdots \\ D_{p36} & (solution\ \ path\, 36\, is\ used) \end{cases} \tag{23}$$

Formula 23 indicates that the values of the decision variables depend on which solution path is used. In other words, by choosing alternative solution paths, we can control the values of the decision variables, which determine the final solution.

In general, let *P* denotes a solution path. When *P* is used, $D_M = D_p$. Let Γ_p denote all possible D_p, i.e. $\Gamma_p = \{D_p\}$. For $D_M \in \Gamma$, we can get the following formula.

$$\Gamma_p \subseteq \Gamma \tag{24}$$

Formula 24 indicates that the different solution paths lead to a large solution space Γ_p, which is a subset of the whole solution space Γ. Like Γ, Γ_p is also a solution space that grows exponentially. For example, suppose *M* is sequentially decomposed into 10 steps, and each step is parallel decomposed into ten alternative methods, then $|\Gamma_p| = 10^{10} = 10{,}000{,}000{,}000$. Theoretically, it is possible to make $\Gamma_p = \Gamma$. But even in the case of $\Gamma_p \subset \Gamma$, Γ_p should contains high quality solutions if $|\Gamma_p|$ is large enough.

Based on this understanding, we can modify formula 11 to get formula 25:

$$\textit{Find}\ \ D_p^* \in \Gamma_p,\ \ \textit{satisfies}\ (5)\ \ \textit{and}\ \ \textit{optimizes}\ (6) \tag{25}$$

From D_p^*, we can get the optimal solution S_p^* in Γ_p with the following formula:

$$S_p^* = \langle F, D_p^* \rangle \tag{26}$$

S_p^* is indeed an approximation of S^*. In practice, it is extremely difficult to get absolute optimal solutions to planning and scheduling problems (NP hard), even idealized theoretical ones. Therefore, we have to adopt certain kinds of approximation methods. When *M* is decomposed properly and Γ_p is large enough, S_p^* should be a good approximation of S^*. With this approximation strategy, we can solve complex decision making problems with an indirect method, i.e., using formula 25 to approximate formula 11.

In summary, with sequential and parallel decompositions, a complex decision making process can be decomposed into simple methods, leading to different solution paths which in turn result in a large solutions space Γ_p. Instead of searching Γ directly, we can search in Γ_p for high quality solutions.

The final solution, and hence its quality, is determined by which solution path is used. Since there are numerous solutions paths, finding one that would lead to a high quality solution quickly becomes an interesting problem. Indeed, the choice of a solution path determines the computation efficiency and solution quality. This will be further discussed in the next section.

A sequential decomposition transforms a complex decision problem into a "multi-stage" decision making process, while a parallel decomposition transforms a decision making problem into an alternative method selection problem. Sequential and parallel decompositions can be used recursively and lead to a "multi-level" decision making scheme. Therefore, this framework is called *multi-stage multi-level decision making* (MSMLDM) model.

The MSMLDM model provides a generic framework to model arbitrary complex decision making processes. It can transform a complex decision making process into a number of simple, small problems that are much easier to solve. The decompositions may lead to a very large solution space, which can be explored accordingly to obtain a high quality solution. As planning and scheduling problems are essentially decision making problems, the MSMLDM model therefore provides a generic framework to formulate complex production planning and scheduling problems.

4.2 The knowledge-directed opportunistic search method

Since the MSMLDM model results in a huge number of solution paths, it is crucial to find a solution path to a high quality solution efficiently. This will determine the computation efficiency and the solution quality. Thus, a knowledge-directed opportunistic search (KDOS) method is developed to find high quality solution paths quickly.

The solution paths form a tree structure. Figure 5 illustrates the solution path tree for the 36 paths of the previous example. Basically, there are three possible strategies to find a high quality solution path from the solution path tree, as follows.

- Try only one path (greedy strategy)
- Try all the paths (brute force strategy)
- Selectively try some of the paths (intelligent strategy)

Take in

Figure 5: The solution path tree

The first strategy is called "greedy" strategy. At each branch point in the tree, the greedy strategy selects only one branch and discards all the rest, based on some local knowledge. Obviously, the computation efficiency of the greedy strategy is excellent. But the myopic and greedy methods generally cannot yield high quality solutions to combinatorial (NP hard) decision making problems.

The second strategy is called “brute force” strategy. Because the brute force strategy traverses the whole solution path tree, it can guarantee to find the optimal solution path. However, its computation time is proportional to the number of solution paths, which grows exponentially and may become extremely large. Very often, the brute force strategy is unacceptably onerous. In other words, its computation efficiency is very bad.

The third strategy is called “intelligent” strategy, which can achieve a good balance between solution quality and computation efficiency.

Essentially, the solution path selection is an AI problem. The greedy strategy relies completely on local knowledge to select the solution path, while the brute force strategy adopts extensive search. Both strategies are impractical, because they cannot meet the solution quality and the computation efficiency requirements simultaneously. As a fundamental law in AI (Fox 1991), *knowledge reduces uncertainty and thereby reducing search, and search compensates lack of knowledge*. In fact, there is no perfect local knowledge that can guarantee optimal (or even good) solutions, but the knowledge can speed up the search and improve the computation efficiency. Indeed, with the intelligent strategy, knowledge-based search methods can be designed to simultaneously meet the requirements on solution quality and computation efficiency, and thus lead to practical methods.

To design an effective method with the intelligent strategy, two AI techniques, namely constraint programming and beam search, are adopted. Constraint programming uses constraints to reduce solution space and thereby improves computation efficiency (Mayoh et al. 1994). From formulas 21 and 22, we can see that sequential decompositions distribute the constraints into different steps. It is possible to check the constraints as early as possible, instead of waiting until a complete solution path is explored. If a constraint is violated, a branch in the solution path tree can be pruned away to reduce the solution space. In formula 22, for example, if constraint C_{111} is violated, then 24 of the 36 solution paths can be dropped, reducing the number of solution paths from 36 to 12. Therefore, with the constraint programming strategy, the computation efficiency can be substantially improved.

Beam search (BS) method selectively explores up to a fixed number (beam width) of branches in the solution tree with some intelligent control strategies (Pinedo 2005). Beam search was first used in artificial intelligence for the speech recognition problem (Lowerre 1976). Fox (Fox and Smith 1984) used beam search for solving complex scheduling problems by a system called ISIS, which is widely recognized as a milestone in both AI and the scheduling research domain. Indeed, it marked the beginning of a new area of scheduling research: *intelligent scheduling*. The beam search method strikes a balance between solution quality and computation efficiency. It perfectly matches the nature of the solution path search problem. Therefore, it is adopted to realize the intelligent strategy.

Prior to discussing the intelligent search method, let’s introduce a solution path tree traversal mechanism. As there are numerous solution paths, a large population of “software robots” are used to explore all the solution paths. We can *create*, *duplicate*, and *destroy* a software robot, which maintains the necessary information for exploring a solution path. At the beginning, there is only one software robot starting from the root of the solution path tree. At each branch point in the solution path tree, a software robot can be duplicated into multiple ones, each of which will continue to explore a different branch. Thus, the population of the software robots will increase with the number of the branches. At the end, every solution path (branch in the solution path tree) is explored by a software robot. This solutions traversal mechanism forms the basis for the intelligent search method.

Figure 6 illustrates the structure of a software robot, which maintains an action stack and the data. The action stack is a dynamic first-in-last-out (FILO) list that stores the actions to be performed. The data maintains the problem structure and solution structure along the solution path.

A software robot accepts three kinds of commands: *push*, *execute*, and *query*. A push command puts a method on the top of the action stack. An execute command pops a method from the action stack and executes the method. A query command queries some information from the robot.

Take in

Figure 6: The structure of a software robot

The execute command can produce four possible results: *false*, *true*, *sequential*, and *parallel*. If the result is *false*, it means that some constraints are violated, suggesting that this branch of solution paths can be pruned away and this software robot can be destroyed. If the result is *true*, it means that a functional method has been successfully executed, and the robot can continue executing. The functional method can modify the robot data with some programming logic. If the result is *sequential*, it means that a composite method is sequentially decomposed. Therefore, a sequence of sub-methods will be pushed onto the action stack. The sub-methods will be executed one by one as a substitute of the composite method. If the result is *parallel*, it means that a composite method is parallel decomposed into a set of alternative methods. Then, the software robot will be duplicated into multiple copies. For each copy, a different alternative method is pushed onto its action stack. Then the software robots will resume executing along different paths. If there is no command remaining in the action stack, the robot finishes exploring a solution path and the robot data provide a solution for the decision making problem. The detailed solution path tree traversal algorithm is specified in table 2 and illustrated in figure 7. The numbers in figure 7 correspond to the numbers in the comments of the pseudo code in table 2.

Take in

Figure 7: The solution path tree traversal mechanism

As the robots are independent of each other, they can be executed in parallel. For simplification, the above traverse algorithm serializes the execution of the robots, and provides a queue $\boldsymbol{S_p}$ to hold the robots that are waiting for execution. The successfully finished robots will enter set $\boldsymbol{S_c}$.

Let's now demonstrate the tree traverse algorithm with the previous example. At the beginning, a robot R_0 is created with input W_0 and method M. Let's represent this as:

$$R_0 = \frac{W_0}{[M]} \tag{27}$$

In formula 27, W_0 is the data of R_0, and $[M]$ means there is only one method M in the robot action stack. Now, when R_0 executes, M is popped out of the action stack to be executed. As M is sequentially decomposed into M_1, M_2, and M_3, the result for executing M is *sequential*, and three sub-methods M_1, M_2, and M_3 are pushed into the action stack. This can be represented as:

$$R_0 = \frac{W_0}{[M]} \xrightarrow{execute} \frac{W_0 + [\dddot{M}]}{[M_1 M_2 M_3]} \tag{28}$$

In formula 28, $[\dddot{M}]$ means M is executed as a sequentially decomposed method, $W_0 + [\dddot{M}]$ represents the current robot data, and $[M_1 M_2 M_3]$ represents the current action stack, which includes three sub-methods. Indeed, $\dddot{M}$ does not change the robot data.

Take in

Table 2: The solution path tree traversal algorithm

Now let's continue to execute the robot R_0. Thus, M_1 is popped out of the action stack to be executed. As M_1 is parallel decomposed into M_{11} and M_{12}, the result for executing M_1 is *parallel*. Then a copy of R_0 is duplicated, namely R_1. Therefore there are two identical robots: R_0 and R_1. M_{11} is pushed into the action stack of R_0, and M_{12} is pushed into the action stack of R_1. Then R_0 and R_1 resume executing along different solution paths. This can be represented as follows.

$$R_0=\frac{W_0}{[M]}\xrightarrow{execute}\frac{W_0+[\dddot{M}]}{[M_1M_2M_3]}\xrightarrow{execute}\begin{cases}\dfrac{W_0+[\dddot{M}\hat{M}_1]}{[M_{11}M_2M_3]}=R_0^{(1)}\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1]}{[M_{12}M_2M_3]}=R_1\end{cases}\tag{29}$$

In formula 29, $\hat{M}_1$ means M_1 is executed as a parallel decomposed method. It also does not change the robot data. Similarly, we can continue executing the robots as follows.

$$R_0^{(1)}\xrightarrow{execute}\frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}]}{[M_{111}M_{112}M_2M_3]}\xrightarrow{execute}\begin{cases}\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}]}{[M_{1111}M_{112}M_2M_3]}=R_0^{(2)}\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}]}{[M_{1112}M_{112}M_2M_3]}=R_2\end{cases}\tag{30}$$

$$\begin{aligned}&R_0^{(2)}\xrightarrow{execute}\frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}]}{[M_{112}M_2M_3]}\xrightarrow{execute}\frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}]}{[M_2M_3]}\\&\xrightarrow{execute}\begin{cases}\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2]}{[M_{21}M_3]}=R_0^{(3)}\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2]}{[M_{22}M_3]}=R_3\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2]}{[M_{23}M_3]}=R_4\end{cases}\end{aligned}\tag{31}$$

In formula 31, $\dot{M}_{1111}$ means M_{1111} is executed as a functional method, which can modify the robot data.

$$\begin{aligned}&R_0^{(3)}\xrightarrow{execute}\frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}]}{[M_3]}\\&\xrightarrow{execute}\frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3]}{[M_{31}M_{32}]}\\&\xrightarrow{execute}\begin{cases}\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}]}{[M_{311}M_{32}]}=R_0^{(4)}\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}]}{[M_{312}M_{32}]}=R_5\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}]}{[M_{313}M_{32}]}=R_6\\[2ex]\dfrac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}]}{[M_{314}M_{32}]}=R_7\end{cases}\end{aligned}\tag{32}$$

$$\begin{aligned} R_0^{(4)} &\xrightarrow{execute} \frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}\dot{M}_{311}]}{[M_{32}]} \\ &\xrightarrow{execute} \frac{W_0+[\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}\dot{M}_{311}\dot{M}_{32}]}{[\quad]} = R_0^{(5)} \end{aligned} \tag{33}$$

Formulas 29 to 33 can be depicted in figure 8. When the software robot R_0 reaches $R_0^{(5)}$, its action stack is empty, thus R_0 finishes exploring a path ($\dddot{M}\hat{M}_1\dddot{M}_{11}\hat{M}_{111}\dot{M}_{1111}\dot{M}_{112}\hat{M}_2\dot{M}_{21}\dddot{M}_3\hat{M}_{31}\dot{M}_{311}\dot{M}_{32}$). As the decomposition methods do not change the robot data, we can omit them and only keep the functional methods, thus we can get the solution path of R_0 ($\dot{M}_{1111}\dot{M}_{112}\dot{M}_{21}\dot{M}_{311}\dot{M}_{32}$). By comparison with the previous formulations of the solution paths, we know that software robot R_0 has explored the solution path P_1. During this process, it has also propagated seven other software robots, namely R_1, R_2, R_3, R_4, R_5, R_6, and R_7. With the same executing mechanism, these software robots can continue executing and propagates other software robots. Finally, all the software robots finish executing and each of them explores a different solution path. Hence all the solution paths are explored accordingly.

Take in

Figure 8: The software robot based solution tree traversal mechanism

Based on the beam search method and the software robot based solution path tree traversal mechanism, let's now discuss how to further develop a computationally efficient method to find high quality solution paths quickly.

The nature of the beam search method is to allocate the computation resources intelligently to a limited number of most promising branches, instead of blindly searching all the paths. As the solution paths are explored by a swarm of software robots, we need to adjust the number of branches indirectly by controlling the population size of the software robots. In other words, we need to allocate the limited computation resources to the most promising software robots.

With the above consideration, a "quota" based computation resource allocation mechanism is designed. As the amount of computation resources depends on the number of branches, we can use the number of branches to measure computation resources. Let Δ denote the total available computation resources in terms branches. In other words, we can explore up to Δ branches. At the beginning, there is only one software robot R_0. Therefore, all the computation resources are allocated to R_0. The computation resources allocated to a software robot is called *quota*. Hence, the quota of R_0 is Δ. This can be represented as follows.

$$\delta(R_0) = \Delta \tag{34}$$

A software robot can propagate a population of software robots to explore multiple branches. Indeed, the quota value is the upper limit of robot population.

At a branch point, a software robot needs to be duplicated into multiple ones to explore different branches. Consequently, the available computation resource of the robot should be split and distributed to different robots. Instead of evenly distributing the computation resources, we can estimate the potential of each branch with some local knowledge, and proportionally allocate computations resources according to the potentials of the branches. Thus the local knowledge can help to minimize potential waste of precious computation resources. This can be formulated as follows.

$$R_i = \underset{j=0}{\overset{n_i}{\{}} R_j \tag{35}$$

$$\delta(R_j) = \frac{\delta(R_i)\rho(R_j)}{\sum_{j=0}^{n_i} \rho(R_j)} \tag{36}$$

Formula 35 means that a robot R_i can be duplicated into n_i robots at a branch point. Formula 36 calculates the quota for each of the duplicated robots. $\rho(R_j)$ means the potential of R_j. $\rho(R_j)$ is calculated with some local knowledge.

Take in

Table 3: Example of quota calculation

Table 3 shows the quota calculation with the previous example problem. As the quota values must be integer numbers, the fractional values are rounded to the nearest integer numbers. The quota values are allocated to the robots from the highest potential to the lowest potential.

In table 3, the quota values for some robots (such as R_3) are zero. This indicates that they do not get any computation resources. Therefore such robots will be destroyed. Indeed, it is not necessary to produce software robots for such branches. Thus, some inferior branches will be pruned off to ensure the computation resources will not exceed the given limit.

With the above computation resource allocation mechanism, the quota values of the robots will decrease with the growth of their population size. When the quota value of a robot becomes one, it cannot produce new robots any longer. Thus, in the remaining part of exploration, the software robot has to completely rely on local knowledge to make greedy decisions on branch selection. This is not desirable for solution quality. For example, suppose a method is sequentially decomposed into 10 steps, each of which can be parallel decomposed into 10 alternative methods as discussed before, then a very large solution space will result. Let Δ=100, the robot population size will grow to 100 after going through the first two steps. In the subsequent 8 steps, each robot continues solution path exploration among 10^8 possible branches, based completely on myopic and greedy decisions.

Now, let's improve the above method with the beam search method. In the previous example, instead of keeping all the robots, we only keep 20 best robots to continue the next step. In other words, the beam width is 20. At the beginning, there is only one robot. After the first step, the robot population is 10, so all of them are retained for the next step. After the second step, the 10 robots may propagate up to 100 robots. Then these robots are evaluated with some local knowledge, 20 of the best robots are retained while all the others are destroyed. The computation resources (totally 100) are now redistributed among the 20 robots according to their performance. These 20 robots continue the third step, after which the robot population grows up to 100 again. These robots are filtered to 20 with the similar evaluation and filter mechanism. This process repeats until the robots have gone through all the 10 steps. In general, we can represent the above mechanism as:

$$\begin{aligned} M &= \sum_{i=0}^{n} A_i \\ A_i &= \begin{cases} M_i \\ E_i \end{cases} \end{aligned} \qquad (37)$$

Formula 37 means a method *M* can be sequentially decomposed into a sequence of actions. Each action A_i can be a method M_i, or an evaluation function E_i. E_i can evaluate the robots with local knowledge to shrink the robot population size and redistribute the computation resources.

Similar to the methods, the evaluation functions can be pushed into the action stack of a robot, and be executed. After an evaluation function is executed, the robot will enter a buffer S_e to wait for the other robots. When all the robots are evaluated and gathered in S_e, they are sorted according to their performance. All inferior robots will be destroyed while good ones are retained, and the computation resources will be redistributed among the good robots. Then the good robots are moved into S_p to be further executed. This can be illustrated with figure 9, which is a modified version of figure 7.

Take in

Figure 9: The knowledge directed opportunistic search

As the decomposition methods can be used recursively, sequential decomposition can occur at multiple levels. Therefore, the above beam search mechanism can be applied in multiple levels accordingly. As the available computation resources vary at different levels, it is not feasible to have a fixed beam width. Hence, filter ratio is introduced. Filter ratio is the ratio of the available computation resources to the number of robots to be retained. This can be represented as follows.

$$r_f = \frac{\delta}{|R|_g} \tag{38}$$

In formula 38, r_f denotes the filter ratio, δ denotes the available computation resources (quota), and $|R|_g$ denotes the number of good robots to be retained. For example, in the previous example, the δ =100, $|R|_g$ =20, therefore r_f =100/20=5. Given the available computation resources and the filter ratio, we can calculate the number of the good robots to be retained with the following formula:

$$|R|_g = \frac{\delta}{r_f} \tag{39}$$

In summary, with the quota based computation resource management method and the beam search mechanism, a swarm of software robots explore a huge solution path tree to find a high quality solution for a complex decision making problem. At each branch point, a software robot can be split into multiple ones, each of which will continue exploring a different branch, the computation resources are distributed to the new generation of robots based on the potential of each branch. Periodically, the software robots are evaluated and inferior ones are pruned off, and the computation resources are redistributed based on the performance of the robots retained. Thus, the population of the robots will experience many rounds of "grow and filter" cycles. In contrast to blind searching, this method tries to minimize the possible waste of computation resources and therefore maximize the possibility of finding high quality solutions with fixed amount of computation resources. As the search process depends largely on the local knowledge, this method is called *knowledge-directed opportunistic search (KDOS).*

The KDOS method uses a fixed amount of computation resources determined by the given parameters Δ and r_f . Its computation efficiency can therefore be guaranteed.

The solution quality of KDOS depends on the given amount of computation resources, the problem decomposition scheme and the quality of the local knowledge. Theoretically, the KDOS method has advantages over the greedy methods and the extensive search methods. Comparing with the greedy methods, the KDOS method can obviously provide better solution quality. This is testified with computation experiments to be presented in section 6. In comparison with the extensive search methods, KDOS should be able to either achieve better solution quality with same amount of computation resources, or achieve the same solution quality with less computation resources, because it can take advantage of constraint programming technique to effectively reduce the search space and local knowledge to minimize potential waste of computation resource. This is to be further verified in future research.

Indeed, the most important advantage of the KDOS method is its *generality*. The generality comes from the generic multi-stage multi-level decision making (MSMLDM) model and the software robot based solution path search mechanism. Both the MSMLDM model and the software robot based search method do not assume any specific problem structure, solution structure, or constraints. Hence, they can be used to solve arbitrary complex decision making problems. This advantage enables the KDOS method to model complex real-world manufacturing systems with multi-agent based distributed decision making techniques, which in turn lead to the multi-agent based beam search (MABBS) method for intelligent production planning and scheduling. This will be further discussed in the next section.

4.3 The multi-agent based beam search method

In general, complex decision making problems are difficult to handle in a centralized approach. For example, it is difficult to model a complex manufacturing system as a monolithic system. Hence, it would be advantageous to decompose a complex system into smaller and simpler units, which cooperate with each other in problem solving and decision making. In other words, for complex decision making problems, the problem structure, solution structure, constraints, and the decision-making and problem-solving knowledge are all distributed among a group of intelligent software agents. This is the nature of multi-agent system (MAS) based scheduling.

Although MAS-based scheduling has many advantages, it essentially uses local knowledge to make myopic and greedy decisions. As a result, MAS-based scheduling usually suffer from poor solution qualities (Shen 2002) (Caridi and Cavalieri 2004). This indeed is a major issue for MAS-based scheduling.

As discussed in the previous section, the KDOS method provides a generic knowledge-based search method that can solve complex decision making problems, while MAS provides an effective method to model complex manufacturing systems. The combination of KDOS and MAS leads to a more powerful method, called *multi-agent based beam search* (MABBS), for effective planning and scheduling of complex real-world manufacturing systems.

The MABBS method can be conceptualized as follows. For a very complex decision making problem, a population of software robots are used to explore the huge solution path tree to find a good solution. The "brain" of each software robot is a multi-agent system. In other words, the intelligence of each software robot comes from a group of intelligent agents.

Software agents and software robots are two different concepts in this paper. Software agents are problem solving and decision-making units, which usually represent some real world counterparts. For example, a resource agent may represent a machine on the shop floor, and manages the schedule of that machine. Software robots are vehicles to probe a large search space. A software robot indeed manages a program instance. It is quite similar to a program process or a program thread.

A multi-agent system is specified as follows.

- A complex dynamic system, such as a manufacturing system, can be modelled with many intelligent software agents, which collaborate with each other to solve complex problems.
- Each software agent has its data and knowledge. A software agent maintains its data with its knowledge.
- Each software agent can request services from other agents, and it can also provide services to the other agents. The agents interact through predefined interfaces.
- Each agent can make decisions with its knowledge in the problem-solving process.

A set of technical measures are used in this research to implement a multi-agent system:

- Each software agent has a unique identifier.
- All the agent data are stored in a common database. An agent can locate its data with its identifier.
- The knowledge of the agents is organized in agent classes. A software agent is an instance of an agent class. Given an agent identifier, we can figure out the agent class, and thus access the knowledge of the agent.
- Each agent class can implement one or more interfaces. Each interface includes one or more problem-solving methods. An agent class can inherit another agent to reuse its knowledge.
- Each software agent belongs to an agent type. Agents of the same type provide services through same interfaces, but they can have different implementations through different agent classes.

In order to integrate the MAS technique into the KDOS framework, we need to solve the following problems.

- How to implement MAS-based distributed decision making?
- How to manage the interactions between the software agents?

- How to manage the data of the software agents?

For the first problem, the MSMLDM model is applied to MAS. Each method of the intelligent agents can be sequentially decomposed or parallel decomposed. The sub-methods can come from the same agent as the owner of the current method, or come from other agents. The decomposition methods can assign the problem solving and decision making tasks to many different software agents, and also lead to a huge solution path tree, which can be explored by the software robots with the KDOS method. The functional methods modify the agent data step by step to reach solutions for complex problems.

For the second problem, two agent interaction methods are designed:

- send request (action stack based)
- post request (message queue based)

Send requests are implemented based on the action stack and they trigger immediate responses, something like making phone calls. Post requests are implemented based on the message queue, something like posting mails. A request in the message queue will not be executed until the receiver agent picks it up from the message queue and pushes it into the action stack. Figure 10 depicts the enhanced software robot structure to support MAS. For each software robot, a message queue and two commands (*post* and *pick*) are introduced to support the post requests. In brief, all the intelligent software agents are executed within the context of a common software robot (like a program process or thread), the action stack and message queue provides flexible mechanisms to control the execution of the software agents.

As shown in figure 10, a method can be pushed in the action stack or posted into the message queue, and it can be specified with four fields of information, namely agent ID, interface name, method name, and method parameters. With such information, the software robot can invoke the agent methods properly.

Take in

Figure 10: The enhanced software robot structure

For the third problem, each software robot provides a database buffer to manage the modified and new added agent data. Although the agent data are stored in database, a software robot cannot directly modify the data in the database, because the database is shared by all the software robots, and different robots may change the agent data in different ways. In order to avoid messing up the data, a "copy on write" strategy is used to manage data modification, as shown in figure 11. This method makes each software robot "feel" as if it owns all the data, and it can modify the data in its private database buffer without affecting other software robots. As the software robots only keep the modified data in the buffer, this will help to reduce memory consumption. It is also convenient to commit the data in the database buffer into the database to save the final solution.

Take in

Figure 11: The copy on write method

In summary, the MAS method and the KDOS method are seamlessly integrated to form a more powerful method. As the KDOS method is based on the beam search method, this method is called multi-agent based beam search (MABBS). The major advantage of the MABBS method is to upgrade the MAS method from local knowledge based myopic and greedy decision makings to knowledge directed intelligent search. This will substantially improve the solution quality of the MAS method. Indeed, the MABBS method can systematically address three major requirements: generality, solution quality, and computation efficiency, within a unified framework. Hence, it provides a promising method for practical production planning and scheduling problems. The fourth requirement (implementation difficulty) will be further discussed in the next section.

5. The MABBS-based open software platform

In order to substantially reduce the implementation difficulty of the MABBS method, an open software platform is developed. As shown in figure 12, the software platform includes five major components:

- the MABBS engine
- the knowledge base
- the database
- the client
- the toolkit

Take in

Figure 12: The MABBS based open software platform

The MABBS engine is the key component of the software platform. It can solve complex decision making problems with the MABBS method. It is designed as a server program, which can receive and handle problem solving requests from the client program. It can create and populate the software robots to perform complex computations and intelligent search, based on the knowledge of intelligent agents. The MABBS method is built into the engine, but the knowledge and data of the intelligent agents are separated from the MABBS engine, and are stored in the knowledge base and the database respectively.

The knowledge base is encoded in a knowledge markup language (KML), which is an XML (Extensible Markup Language) based markup language to organize the knowledge of the intelligent agents with agent classes and interfaces. A script language, called embedded extensible application script language (EXASL) is developed to program the methods of the agent classes. EXASL is embedded in KML, which can be loaded and executed by the MABBS engine. With built-in XML and database functions, EXASL can manipulate database data and XML-based complex data structures conveniently. More importantly, EXASL makes it very convenient to specify sequential and parallel decompositions of the agent methods. In summary, KML and EXASL provide a flexible and productive programming facility to develop a multi-agent based knowledge base for complex decision making problems.

SQL (Structured Query Language) based database are used to store the agent data. Firstly, this simplifies the agent data management (as discussed in the previous “copy on write” mechanism). Secondly, the SQL language and relational database are mature and standard data management techniques, this also makes it convenient to share data and integrate with other software systems.

The client program provides user friendly graphical user interfaces (GUI) for effective user interactions. It is also a very important part for a production planning and scheduling system (Pinedo 2002). A smart client program is developed, and EXASL can also be used to develop the GUIs. The fact that both the server side and the client side use the common programming language can greatly reduce the complexity of the software platform as well as the effort on application system development.

A toolkit is developed to facilitate the development of knowledge base and GUI. Figure 13 shows a screenshot of the toolkit. It provides an integrated environment for editing and debugging the programs. A syntax sensitive EXASL editor is developed based on Scintilla (Scintilla 2007), which is an open-source source code editor.

In summary, the open software platforms provide a total solution for implementing practical production planning and scheduling systems. With the software platform, the knowledge engineers can focus on the knowledge of the intelligent agents, instead of the implementation of the search process and low-level implementation details. An operation system (OS) can substantially reduce the difficulty and effort on application program development. Similarly, the open software platform provides reusable infrastructure and flexible tools, which can greatly reduce the implementation difficulty of MABBS method for complex real world problems.

Take in

Figure 13: The toolkit program for knowledge base development

6. The prototype system and computation experiments

In order to verify the feasibility and effectiveness of this research, a prototype production planning, scheduling and executing system was developed based on the requirements of a local manufacturing company, and preliminary computation experiments were conducted to evaluate the performance of the MABBS method.

A manufacturing system is modelled as an intra-organization supply chain based on the bill of materials (BOM) and production routes. Four types of software agents, *planning agent*, *scheduling agent*, *operation manager agent*, and *resource agent* are developed for distributed planning and scheduling, as illustrated in figure 14.

The planning and scheduling process can be described as follows. When a production order (which prescribes what product and what quantity to produce) enters the manufacturing system, the planning agent decomposes the production order according to the BOM, and produces a detailed process plan according to the production routes. There may be multiple alternative production routes, the planning agent need to make decisions on production route selection. Then the scheduling agent allocates resources for the production order according to the process plan produced by the planning agent. The scheduling agent can make decisions on different scheduling policies. As there are different operations in a production plan and different operations may have different constraints and requirements, a group of operation manager agents are created to manage the different operations. When the scheduling agent allocates resource for operations, it passes the requests to the corresponding operation manager agents. Each operation manager agent manages a type of operations, which may be processed on one or more applicable production resources. Each resource is managed by a resource agent. The operation manager agent assigns the operations to the resource agents with a bid mechanism. Each resource agent maintains the schedule of a production resource (such as a production line, or a machine). Different resources may have different constraints and different resource allocation policies. Thus the planning and scheduling are merged into an integrated MAS based problem solving and decision making process.

Take in

Figure 14: The MAS model for production planning and scheduling

Take in

Figure 15: Screenshots of the prototype system

Some complicated realistic factors are taken into consideration in the production planning and scheduling process, including the following:

- Alternative production routes selection;
- Part-production operations and assembly operations;
- Capacity constraints and material constraints;
- Priority and due dates for the production orders;
- Changeover-dependent setup time;
- Shift patterns and factory holidays;
- Resource preferences and production route preferences;
- Overlap between operations;
- Multiple optimization objectives, including make span, tardiness, and setup time.

A workflow-based shop floor data collection system is integrated with the planning and scheduling function to keep the production schedule synchronized with the shop floor executing status in real-time. Figure 15 shows two screenshots of the prototype system.

The prototype system is based on a realistic manufacturing system configuration and a number of factors can be handled in the planning and scheduling process. Its successful development demonstrates the generality of the MABBS method and the feasibility of the software platform. Indeed, comparing with the extensive search methods, generality is the major advantage of the

MABBS method, as it can handle complex practical problems that are often difficult for most extensive search methods.

On the other hand, as discussed in section 4, the MABBS method can provide better solution quality than the greedy methods, such as the MAS based scheduling, expert system scheduling, etc. This can be testified with the following method. As the greedy approach only tries one solution path, we can emulate the greedy approach by setting Δ=1. If Δ=1, there will be only one software robot and it will not produce new software robots. Therefore the software robot has to make greedy and myopic decisions in the problem solving process. Keeping everything the same while changing only the value of Δ, we can compare the performance of normal MABBS method (Δ>1) and the greedy method (Δ=1). The computation experiment results are plotted in figure 16.

Take in
Figure 16: The relationship between solution quality and computation resources of MABBS method

The curves in figure 16 reflect the trend of solution quality with the increase of computation resources. Relative solution quality (0-100) is used for comparison. It is converted from a composite optimization objective to simultaneously minimize the make span, tardiness penalty, and setup time. A greater value of relative solution quality means better solution quality. The three curves are based on three independent sets of test data. From the curves, we can see that the greedy decision approach (Δ=1) gets the worst solution quality for all the three independent experiments, and the solution quality improves with the increasing of the computation resources. This testifies that the MABBS method can provide better solution quality than the greedy methods. The details of the computation experiments are available in (Kang 2007).

Indeed, we are continuously improving the system and trying to put the system into operation in a real manufacturing environment. This will further testify the effectiveness of this research.

7. Conclusions and future work

Production planning and scheduling problems are complex decision making problems of great practical value. Most research results are not practicable for real world problems. In order to narrow this gap, four major requirements, generality, solution quality, computation efficiency and implementation difficulty are indentified for development of practical production planning and scheduling methods. Accordingly, a set of methods and tools are developed based on the four requirements to provide a practical solution for complex real-world production planning and scheduling problems. This study can be summarized as in figure 17. Firstly, a generic multi-stage multi-level decision making (MSMLDM) model is developed to formulate complex decision making processes. Secondly, by combining the MSMLDM model and the beam search method, a software-robot based knowledge directed opportunistic search (KDOS) method is developed to efficiently find high quality solutions for complex decision making problems. Thirdly, by seamlessly integrating the KDOS method with the multi-agent system (MAS) technique, a more powerful method, multi-agent based beam search (MABBS) method is developed for complex real-world production planning and scheduling problems. Fourthly, an open software platform and computer languages (KML, EXASL) are developed to reduce the implementation difficulty of the MABBS method. Finally, a prototype production planning, scheduling and executing system is developed to apply the MABBS method in a real industrial environment. The feasibility and generality of the MABBS method is demonstrated by the prototype system. The advantage of the MABBS method over the greedy methods is testified with computation experiments.

As the MABBS method systematically considers all the four major requirements, it provides a promising method toward practical production planning and scheduling. The MABBS method provides a generic knowledge based search framework, and the knowledge base can be organized in a MAS-based distributed approach.

Take in

Figure 17: Summary of this study

Indeed, more work would be needed to apply the MABBS method into industrial practice and to investigate how to design the multi-agent based knowledge base and how to acquire useful knowledge effectively. Although the development of the MABBS method is motivated by production planning and scheduling of manufacturing systems, its generality and the open software platform render it possible to apply the MABBS method to other complex decision making problems. It would be valuable to further improve the MABBS method and the software platform for broader applications to real-world problems.

References

Adams, J., Balas, E. and Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. Management Science, 34 (3), 391–401.
Amico, M.D. and Trubian, M., 1993. Applying tabu search to the job-shop scheduling problem. Annals of Operations Research, 41 (1–4), 231–252.
Bellifemine, F., Poggi, A. and Rimassa, G., 2001. Developing multi-agent systems with a FIPA compliant agent framework. Software: Practice and Experience, 31 (2), 103–128.
Bruno, G., Elia, A. and Laface, P., 1986. A rule-based system to schedule production. IEEE Computer, 19 (7), 32–40.
Bruno, G. and Morisio, M., 1987. The role of rule based programming for production scheduling. In: Proceedings of IEEE international conference on robotics and automation, Rayleigh, NC, 4, 545–550.
Caridi, M. and Cavalieri, S., 2004. Multi-agent systems in production planning and control: an overview. Production Planning & Control, 15 (2), 106–118.
Dangelmaier, W., Heidenreich, J. and Pape, U., 2005. Supply chain management: a multi-agent system for collaborative production planning. In: Proceedings of the 2005 IEEE international conference on e-Technology, e-Commerce and e-Service (EEE’05), 29 March–1 April, Hong Kong. Washington, DC: IEEE Computer Society, 309–314.
Fox, M.S. and Smith, S.F., 1984. ISIS – a knowledge-based system for factory scheduling. Expert Systems, 1 (1), 25–47.
Fox, M.S., 1991. Introduction to artificial intelligence and expert systems. (IEEE). Video recording, ISBN 0879425475.
Fox, M.S., Barbuceanu, M. and Teigen, R., 2000. Agent-oriented supply-chain management. International Journal of Flexible Manufacturing Systems, 12 (2–3), 165–188.
Harmonosky, C.M., 1995. Simulation-based real-time scheduling: review of recent developments. In: Proceedings of the 1995 winter simulation conference, Arlington, Virginia, USA, 220–225.
Herrmann, J.W., 2004. Information flow and decision-making in production scheduling. In: Proceedings of the 2004 industrial engineering research conference, 15–19 May 2004, Houston, Texas.
Herrmann, J.W., 2005. A history of decision – making tools for production scheduling. In: Proceedings of the 2nd multidisciplinary international conference on scheduling theory and applications (MISTA), 18–21 July 2005, New York, USA, 2, 380–389.
Jacobs, F.R. and Weston Jr, F.C., 2007. Enterprise resource planning (ERP) – a brief history. Journal of Operations Management, 25 (2), 357–363.
Jain, A.S. and Meeran, S., 1999. Deterministic job-shop scheduling: past, present and future. European Journal of Operational Research, 113 (2), 390–434.
Jensen, M.T., 2003. Generating robust and flexible job shop schedules using genetic algorithms. IEEE Transactions on Evolutionary Computation, 7 (3), 275–288.
Kang, S., 2007. Multi-agent based beam search for intelligent production planning and scheduling. Thesis (PhD). Hong Kong: The University of Hong Kong.
Kobayashi, S., Ono, I. and Yamamura, M., 1995. An efficient genetic algorithm for job shop scheduling problems. In: Proceedings of sixth international conference on genetic algorithms, Morgan Kaufmann, 506–511.
Kouiss, K., Pierreval, H. and Mebarki, N., 1997. Using multi-agent architecture in FMS for dynamic scheduling. Journal of Intelligent Manufacturing, 8 (1), 41–47.
Lageweg, B.J., Lenstra, J.K. and Rinnooy Kan, A.H.G., 1977. Jobshop scheduling by implicit enumeration. Management Science, 24 (4), 441–450.
Lowerre, B.T., 1976. The HARPY speech recognition system. Pittsburgh, PA: Carnegie Mellon University.
Marcus, R., 1990. Simulation-based planning, scheduling, and control of job shops. CIM-Review, 7 (1), 44–49.
Mayoh, B.H., Tyugu, E.K. and Penjam, J., 1994. Constraint programming (NATO ASI). Berlin: Springer-Verlag.
Mönch, L. and Stehli, M., 2006. ManufAg: a multi-agent-system framework for production control of complex manufacturing systems. Information Systems and e-Business Management, 4 (2), 159–185.

Mönch, L., et al., 2006. The FABMAS multi-agent-system prototype for production control of water fabs: design, implementation and performance assessment. Production Planning & Control, 17 (7), 701–716.
Musselman, K., O'Reilly, J. and Duket, S., 2002. The role of simulation in advanced planning and scheduling. In: Proceedings of winter simulation conference, 8–11 December 2002, San Diego, CA, 1825–1830.
Myers, K. and Smith, S., 1999. Issues in the integration of planning and scheduling for enterprise control. In: Proceedings of the DARPA symposium on advances in enterprise control, November 1999, San Diego, CA.
O'Brien, P.D. and Nicol, R.C., 1998. FIPA – towards a standard for software agents. BT Technology Journal, 16 (3), 51–69.
Ow, P.S. and Morton, T.E., 1988. Filtered beam search in scheduling. International Journal of Production Research, 26 (1), 35–62.
Pinedo, M., 2002. Scheduling: theory, algorithms, and systems. Upper Saddle River, NJ: Prentice Hall.
Pinedo, M., 2005. Planning and scheduling in manufacturing and services. New York: Springer.
Sabuncuoglu, I. and Bayiz, M., 1999. Job shop scheduling with beam search. European Journal of Operational Research, 118 (2), 390–412.
Scintilla, 2007. Available from: http://www.scintilla.org/ [Accessed 2 January 2007].
Shen, W., 2002. Distributed manufacturing scheduling using intelligent agents. IEEE Intelligent Systems, 11 (7), 88–94.
Shen, W., et al., 2006. Applications of agent-based systems in intelligent manufacturing: an updated review. Advanced Engineering Informatics, 20 (4), 415–431.
Smith, R.G., 1980. The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29 (12), 1104–1113.
Smith, S.F., 1992. Knowledge-based production management: approaches, results and prospects. Production Planning & Control, 3 (4), 350–380.
Smith, S.F., 2003. Is scheduling a solved problem? In: Proceedings of first multidisciplinary international conference on scheduling: theory and applications (MISTA), 3–18 August 2003, Nottingham, UK.
Tan, W. and Khoshnevis, B., 2000. Integration of process planning and scheduling – a review. Journal of Intelligent Manufacturing, 11 (1), 51–63.
Van Laarhoven, P.J.M., Aarts, E.H.L. and Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. Operations Research, 40 (1), 113–125.
Wang, S.-J., Xia, L.-F. and Zhou, B.-H., 2008. FBS-enhanced agent-based dynamic scheduling in FMS. Engineering Applications of Artificial Intelligence, 21 (4), 644–657.
Yang, F.Y., 2003. Machine-order search space for job-shop scheduling. Thesis (PhD). The University of Hong Kong.

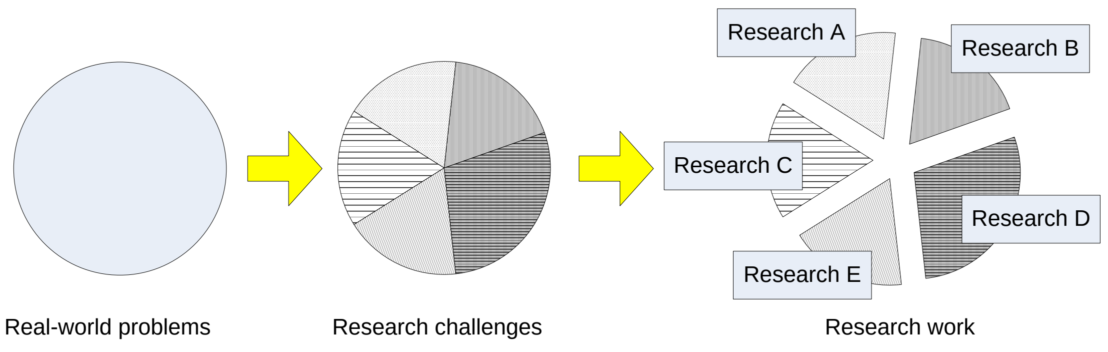


Figure 1: Traditional research methodology

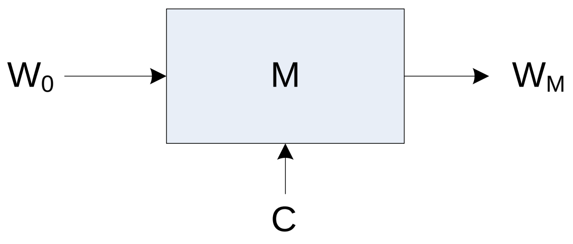


Figure 2: Illustration of a planning and/or scheduling method

Table 1: Definition of planning and scheduling problems with the generic framework

Scheduling	P_0	Allocate resources for the jobs
	S_0	Empty schedule
	C	Capacity constraints, precedence constraints, etc.
	S_M	The final schedule
	o	Make span, tardiness, setup time, etc.
Re-scheduling	P_0	Handle some changes
	S_0	Existing schedule
	C	Capacity constraints, precedence constraints, etc.
	S_M	The revised schedule
	o	Make span, tardiness, setup time, stability, etc.
Planning	P_0	Create detailed process plan for some production tasks
	S_0	Empty plan
	C	Operation sequence, geometric constraints, etc.
	S_M	The production process plan
	o	Production cost, etc.

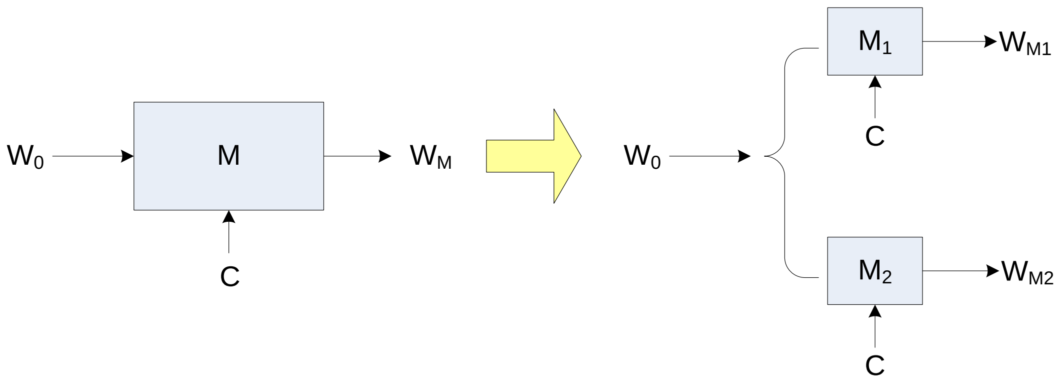


Figure 3: Parallel decomposition

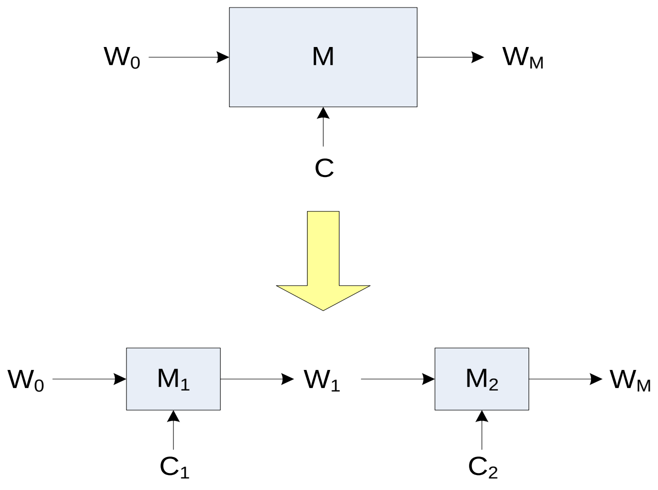


Figure 4: Sequential decomposition

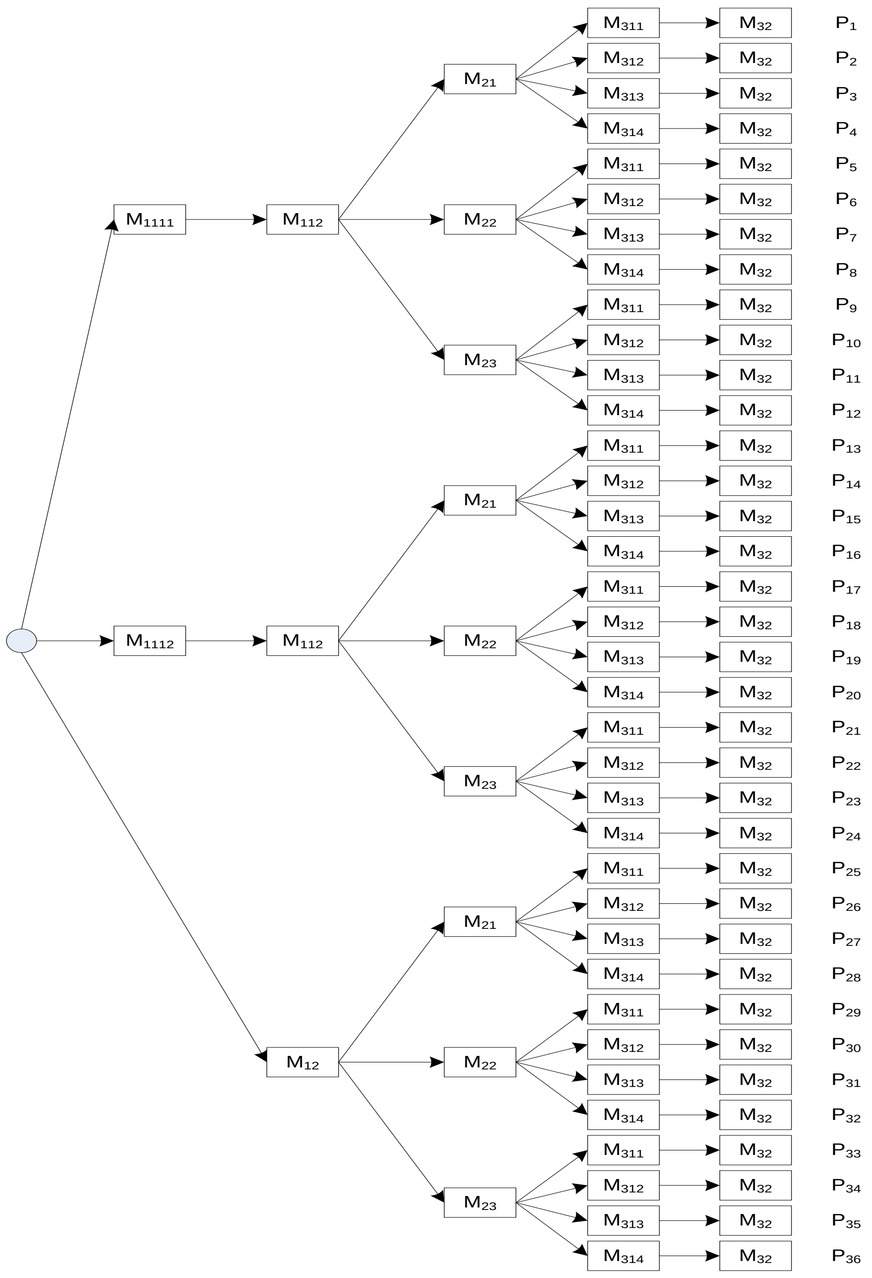


Figure 5: The solution path tree

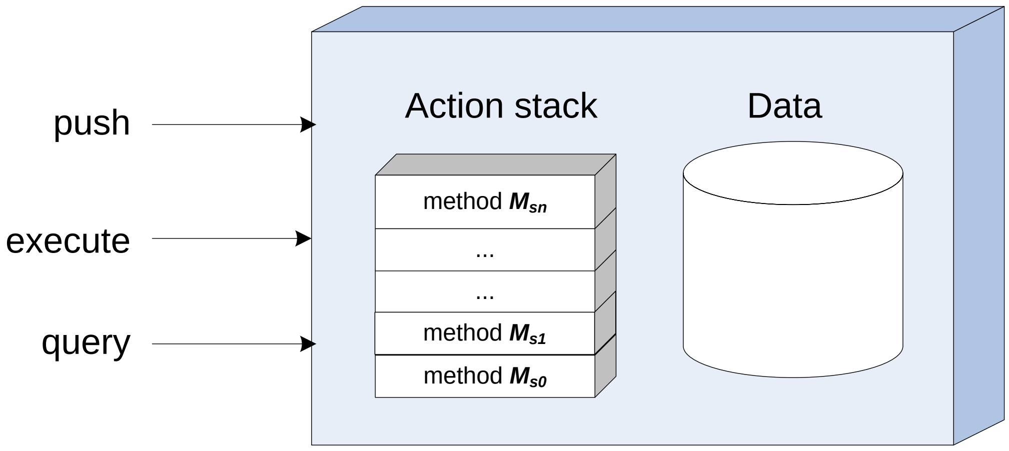


Figure 6: The structure of a software robot

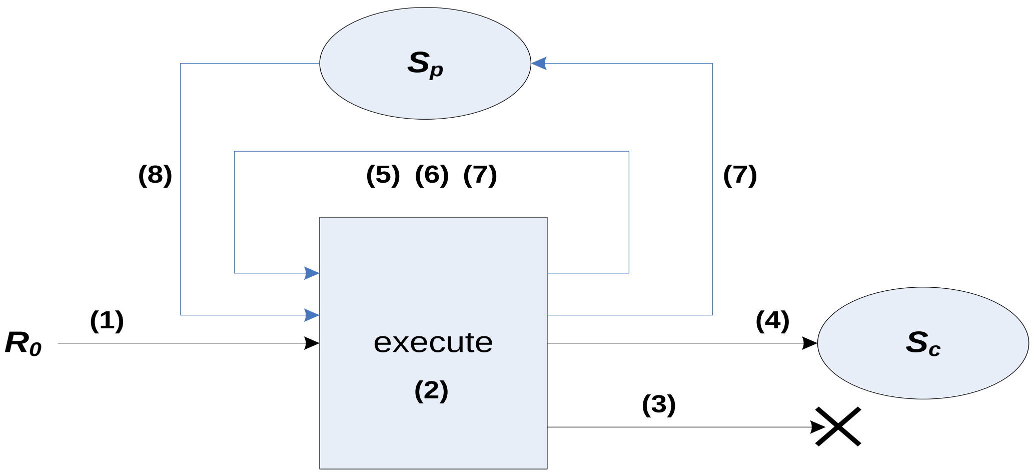


Figure 7: The solution path tree traversal mechanism

Table 2: The solution path tree traversal algorithm

```
    /* (1) initialize */
    $R_0$=create($W_0$, $M$);
    Let $R$=$R_0$;
continue:
      /* (2) execute method on the stack top */
      Let result=$R$.execute();
      /* (3) violate constraints */
      if (result==false) {
            destroy($R$);
            goto next;
      }
      /* method successfully executed */
      if(result==true) {
            if($R$.query(finished)) {
                  /* (4) arrive goal state */
                  add $R$ into $S_c$;
                  goto next;
            }
            /* (5) continue working */
            goto continue;
      }
      /* (6) method sequentially decomposed into $M_{d0}$ ... $M_{dn}$*/
      if(result==sequential) {
            for ($M_d$=$M_{dn}$ to $M_{d0}$) R.push($M_d$);
            goto continue;
      }
      /* (7) method parallel decomposed into $M_{d0}$...$M_{dn}$ */
      if(result==parallel) {
            for ($M_d$=$M_{d1}$ to $M_{dn}$) {
                  Let $R_d$=duplicate($R$);
                  $R_d$.push($M_d$);
                  add $R_d$ into $S_p$;
            }
            $R$.push($M_{d0}$);
            goto continue;
      }
next:
      if(|$S_p$|>0) {
            /* (8) pick up the next robot and continue */
            Let $R$=first element of $S_p$;
            $S_p$= $S_p$-$R$;
            goto continue;
      }
```

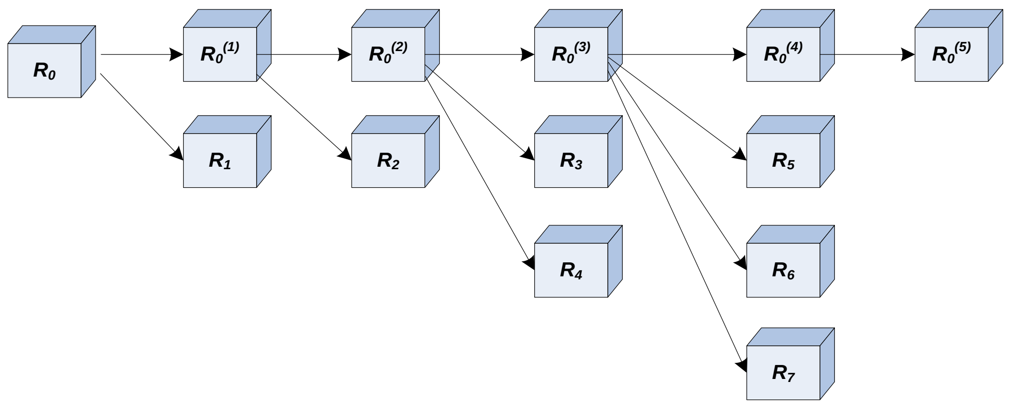


Figure 8: The software robot based solution tree traversal mechanism

Table 3: Example of quota calculation

R_i	R_j	$\rho(R_j)$	$\delta(R_j)$ ($\delta(R_i)$=15)		$\delta(R_j)$ ($\delta(R_i)$=5)		$\delta(R_j)$ ($\delta(R_i)$=3)	
$R_0^{(2)}$	$R_0^{(3)}$	50	5	5	5/3	2	1	1
	R_3	20	2	2	2/3	0	2/5	0
	R_4	80	8	8	40/15	3	24/15	2

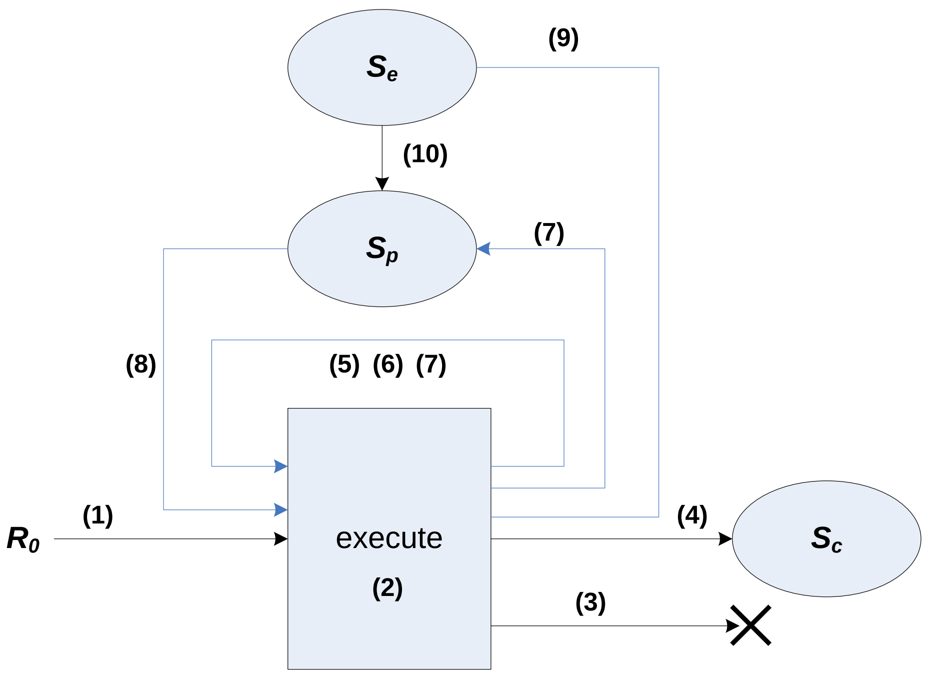


Figure 9: The knowledge directed opportunistic search

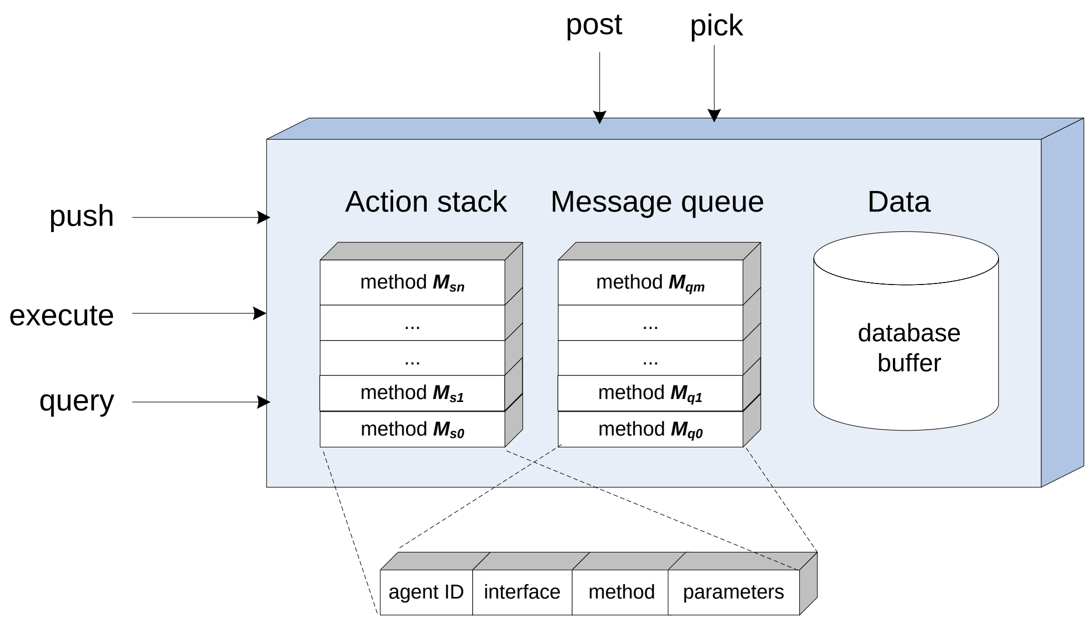


Figure 10: The enhanced software robot structure

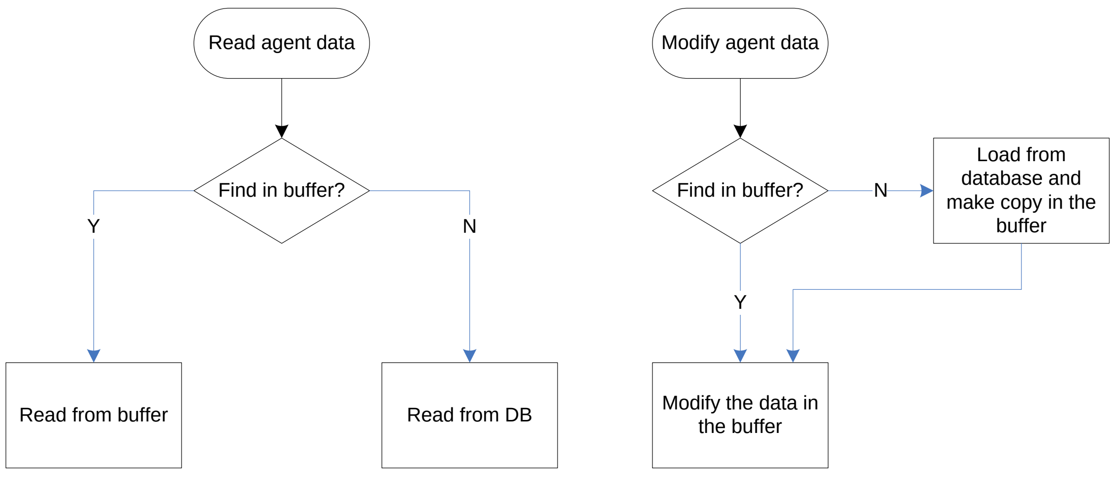


Figure 11: The copy on write method

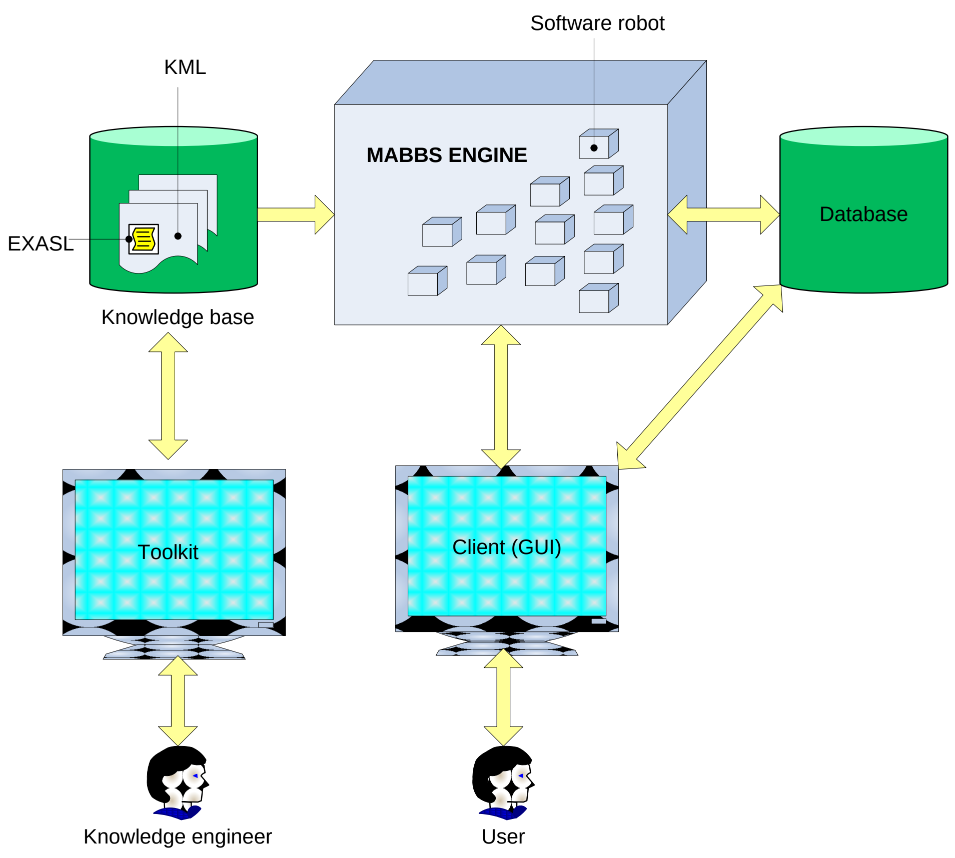
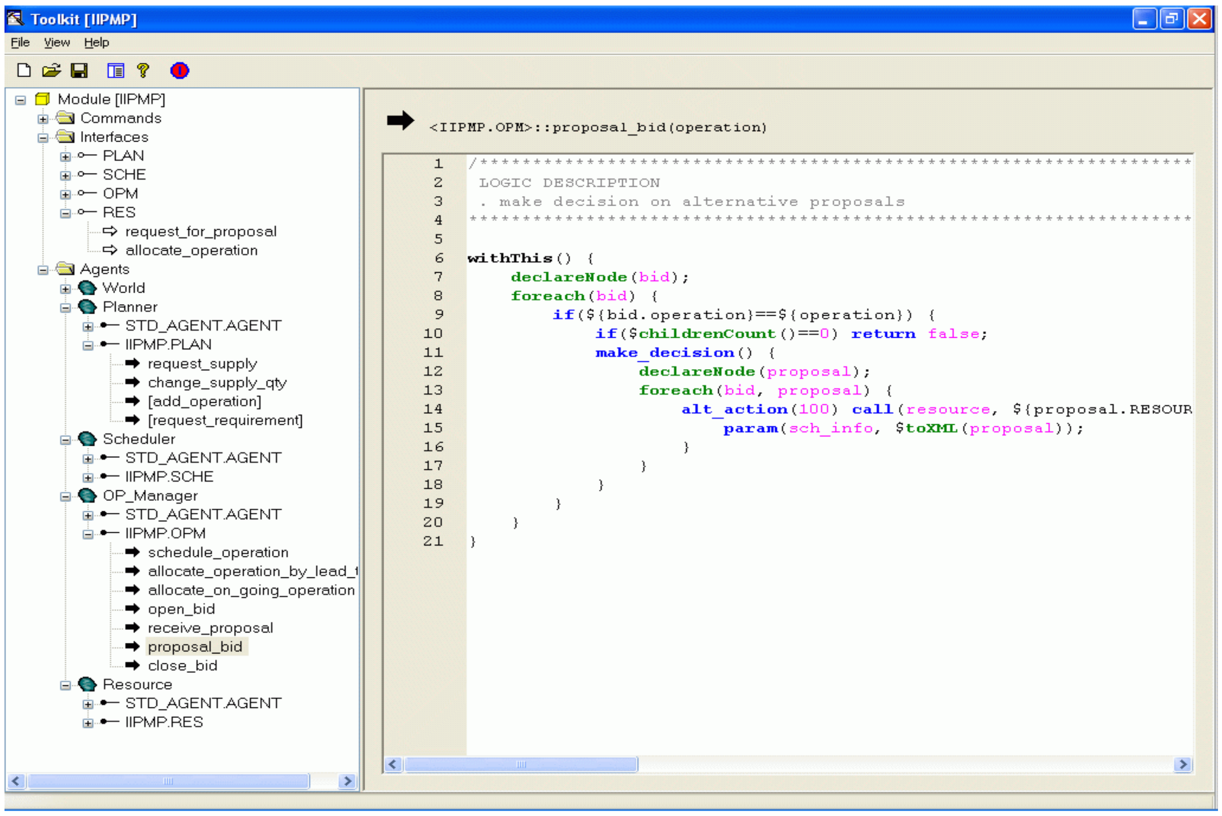


Figure 12: The MABBS based open software platform



```
<IIPMP.OPM>::proposal_bid(operation)
```

```
/*******************************************************************
 LOGIC DESCRIPTION
 . make decision on alternative proposals
*******************************************************************

withThis() {
    declareNode(bid);
    foreach(bid) {
        if(${bid.operation}==${operation}) {
            if($childrenCount()==0) return false;
            make_decision() {
                declareNode(proposal);
                foreach(bid, proposal) {
                    alt_action(100) call(resource, ${proposal.RESOUR
                        param(sch_info, $toXML(proposal));
                    }
                }
            }
        }
    }
}
```

Figure 13: The toolkit program for knowledge base development

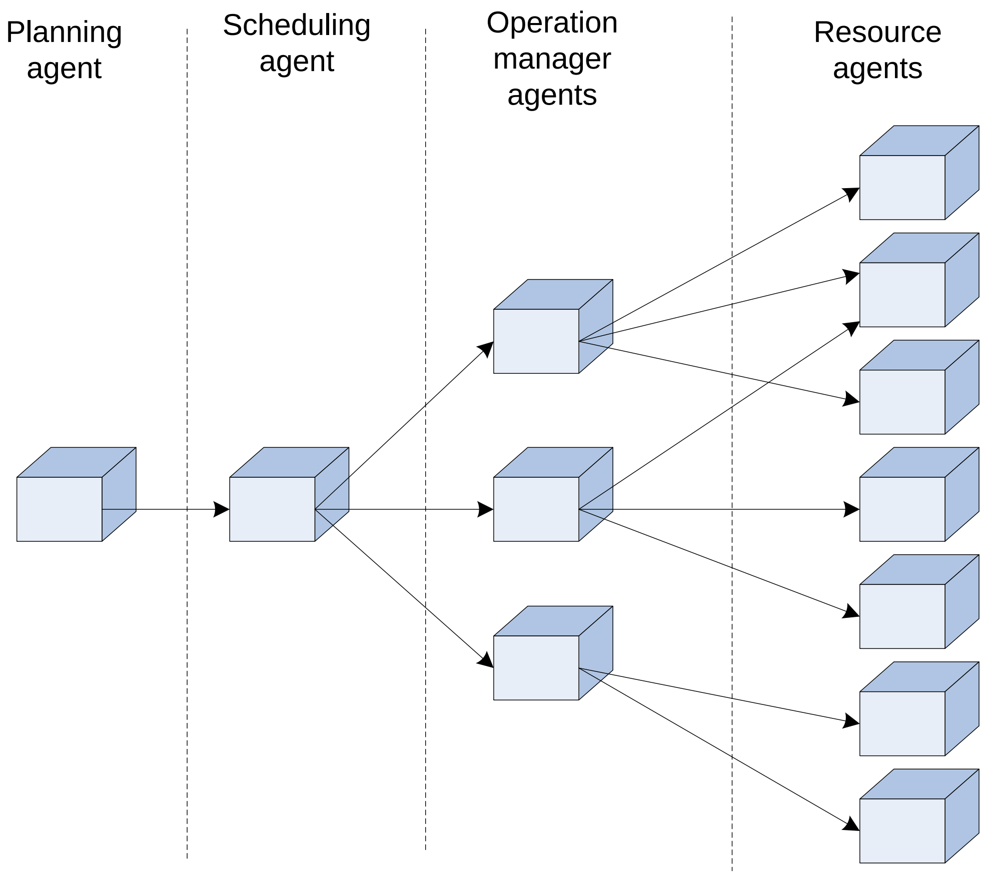
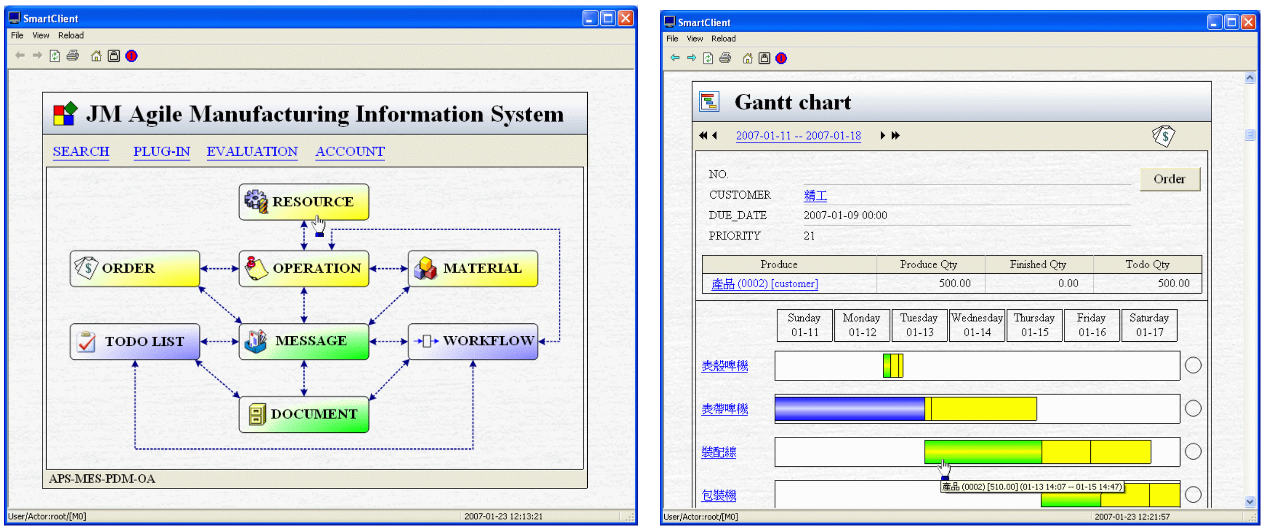


Figure 14: The MAS model for production planning and scheduling



NO.	
CUSTOMER	精工
DUE_DATE	2007-01-09 00:00
PRIORITY	21

Produce	Produce Qty	Finished Qty	Todo Qty
產品 (0002) [customer]	500.00	0.00	500.00

Sunday 01-11	Monday 01-12	Tuesday 01-13	Wednesday 01-14	Thursday 01-15	Friday 01-16	Saturday 01-17

表殼啤機
表帶啤機
裝配線
包裝機

產品 (0002) [510.00] (01-13 14:07 -- 01-15 14:47)

Figure 15: Screenshots of the prototype system

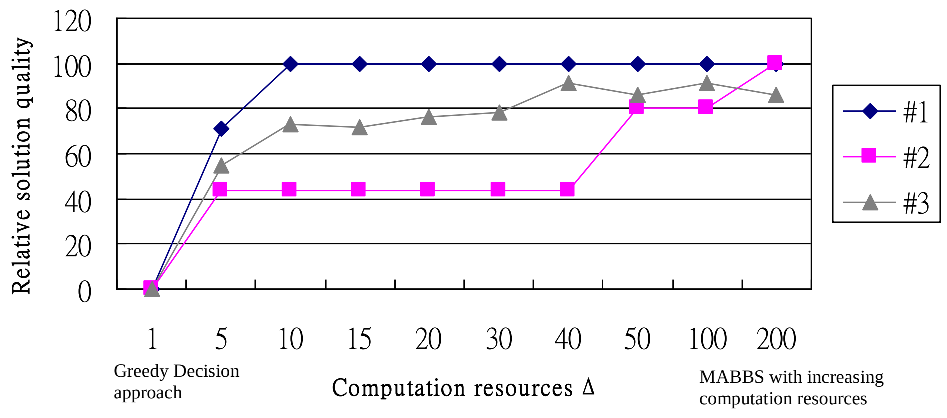


Figure 16: The relationship between solution quality and computation resources of MABBS method

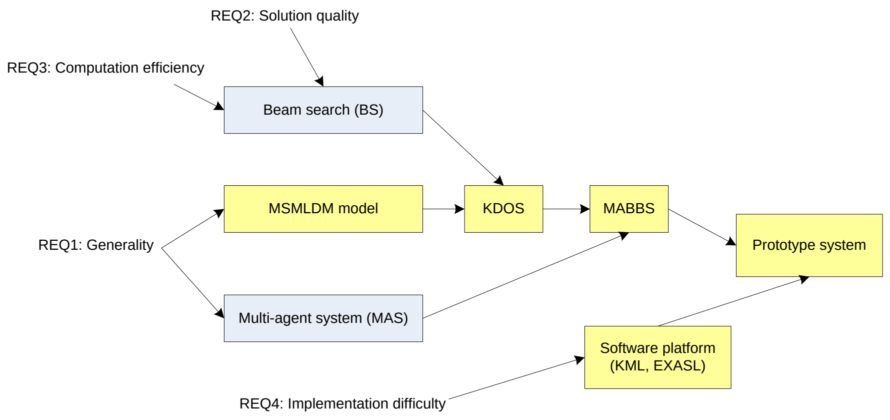


Figure 17: Summary of this study