



## 2. Resource Allocation Framework

First, a deterministic resource allocation problem is considered: an instance of the problem can be characterized by an 8-tuple  $\langle \mathcal{R}, \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{C}, d, e, i \rangle$ . In detail the problem consists of a set of reusable *resources*  $\mathcal{R}$  together with  $\mathcal{S}$  that corresponds to the set of possible *resource states*. A set of allowed *operations*  $\mathcal{O}$  is also given with a subset  $\mathcal{T} \subseteq \mathcal{O}$  which denotes the *target operations* or *tasks*.  $\mathcal{R}$ ,  $\mathcal{S}$  and  $\mathcal{O}$  are supposed to be finite and they are pairwise disjoint. There can be *precedence constraints* between the tasks, which are represented by a partial ordering  $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ . The *durations* of the operations depending on the state of the executing resource are defined by a *partial* function  $d : \mathcal{S} \times \mathcal{O} \rightarrow \mathbb{N}$ , where  $\mathbb{N}$  is the set of natural numbers, thus, we have a discrete-time model. Every operation can *effect* the state of the executing resource, as well, that is described by  $e : \mathcal{S} \times \mathcal{O} \rightarrow \mathcal{S}$  which is also a partial function. It is assumed that  $\text{dom}(d) = \text{dom}(e)$ , where  $\text{dom}(\cdot)$  denotes the domain set of a function. Finally, the *initial states* of the available resources are given by  $i : \mathcal{R} \rightarrow \mathcal{S}$ .

The state of a resource can contain all relevant information about it, for example, its type and current setup (scheduling problems), its location and load (logistic problems) or its condition (maintenance and repair problems). Similarly, an operation can effect the state in many ways, e.g., it can change the setup of the resource, its location or its condition. The system must allocate each task (target operation) to a resource, however, there may be cases when first the state of a resource must be modified in order to be capable of executing a certain task (e.g. a transporter may first need to travel to its loading/source point, a machine may require repair or setup). In these cases non-task operations can be applied. They can modify the states of the resources without directly serving a demand (executing a task). It may be the case that during the resource allocation process a *non-task* operation is applied several times, but other *non-task* operations are completely avoided (for example, because of their high cost). Nevertheless, finally all *tasks* must be completed.

A *solution* for a deterministic RAP is a partial function, the *resource allocator function*,  $\varrho : \mathcal{R} \times \mathbb{N} \rightarrow \mathcal{O}$  that assigns the *starting times* of the operations on the resources. Note that the operations are supposed to be *non-preemptive* (they may not be interrupted).

A solution to a RAP is called *feasible* if and only if the following four properties are satisfied:

- (1) Each task is rendered to exactly one resource and start time:  $\forall v \in \mathcal{T} : \exists! \langle r, t \rangle \in \text{dom}(\varrho) : v = \varrho(r, t)$
- (2) All resources execute at most one operation at a time:  $\neg \exists u, v \in \mathcal{O} : u = \varrho(r, t_1) \wedge v = \varrho(r, t_2) \wedge t_1 \leq t_2 < t_1 + d(s(r, t_1), u)$
- (3) The precedence constraints of the tasks are kept:  $\forall \langle u, v \rangle \in \mathcal{C} : [u = \varrho(r_1, t_1) \wedge v = \varrho(r_2, t_2)] \Rightarrow [t_1 + d(s(r_1, t_1), u) \leq t_2]$
- (4) Every operation-to-resource assignment is valid:  $\forall \langle r, t \rangle \in \text{dom}(\varrho) : \langle s(r, t), \varrho(r, t) \rangle \in \text{dom}(d)$

where  $s : \mathcal{R} \times \mathbb{N} \rightarrow \mathcal{S}$  describes the states of the

resources at given time points, and it is defined as

$$s(r, t) = \begin{cases} i(r) & \text{if } t = 0 \\ s(r, t-1) & \text{if } \langle r, t \rangle \notin \text{dom}(\varrho) \\ e(s(r, t-1), \varrho(r, t)) & \text{otherwise} \end{cases}$$

A RAP is called *correctly specified* if there exists at least one feasible solution. In what follows it is assumed that the problems are correctly specified. The set of all feasible solutions is denoted by  $\mathbb{S}$ . There is a performance (or cost) associated with each solution defined by a *performance measure*  $\kappa : \mathbb{S} \rightarrow \mathbb{R}$  that often depends on the task completion times, only. Typical performance measures that appear in practice include: maximum completion time or mean flow time. The aim of resource allocation is to compute a feasible solution with maximal performance (or minimal cost).

Note that the performance measure can assign penalties for violating *release* and *due* dates (if they are available) or can even reflect the *priority* of the tasks.

So far our model was deterministic, now we turn to stochastic RAPs. The stochastic variant of the described general class of RAPs can be defined by randomizing functions  $d$ ,  $e$  and  $i$ . Consequently, the operation durations become random,  $d : \mathcal{S} \times \mathcal{O} \rightarrow \Delta(\mathbb{N})$ , where  $\Delta(\mathbb{N})$  is the space of probability distributions over  $\mathbb{N}$ . The effect of the operations are also uncertain,  $e : \mathcal{S} \times \mathcal{O} \rightarrow \Delta(\mathcal{S})$  and the initial states of the resources can be stochastic, as well,  $i : \mathcal{R} \rightarrow \Delta(\mathcal{S})$ . Note that the elements in the domain sets of functions  $d$ ,  $e$  and  $i$  are probability distributions, we denote the corresponding random variables by  $D$ ,  $E$  and  $I$ , respectively. We use the notation  $X \sim f$  to indicate that random variable  $X$  has probability distribution  $f$ . Thus,  $D(s, o) \sim d(s, o)$ ,  $E(s, o) \sim e(s, o)$  and  $I(r) \sim i(r)$  for all  $s \in \mathcal{S}$ ,  $o \in \mathcal{O}$  and  $r \in \mathcal{R}$ . In stochastic RAPs the performance of a solution is also a random variable. Therefore, in order to compare the performance of different solutions we have to compare random variables. There are many ways in which this comparison can be made. For example, we can say that a random variable has stochastic dominance over another random variable "almost surely", "in likelihood ratio sense", "stochastically", "in the increasing convex sense" or "in expectation". In different applications various types of comparisons can be suitable, however, probably the most natural one is based upon the expected values of the random variables. The paper applies this kind of comparison.

Now, we classify the basic types of resource allocation techniques. In deterministic RAPs, there is no real difference between open- and closed-loop control. In that case, we can safely restrict ourself to open-loop methods. If the solution is aimed at generating the resource allocation off-line in advance, then it is called *predictive*. Thus, predictive solutions perform open-loop control and assume a deterministic environment. In stochastic resource allocation there are some data (e.g., the actual durations) that will only be available during the execution of the plan. According to the usage of these information, we identify two basic types

of solution techniques. An open-loop solution that can deal with the uncertainties of the environment is called *proactive*. A proactive solution allocates the operations to resources and defines the orders of the operations, but, because the durations are uncertain, it does not determine precise starting times. This kind of technique can be applied when only the durations of the operations are stochastic, but, the states of the resources are known perfectly (e.g. stochastic job-shop scheduling).

Finally, in the stochastic case a closed-loop solution to a RAP is called *reactive*. A reactive solution is allowed to make the decisions on-line, as the resource allocation process actually evolves and more information becomes available. Naturally, a reactive solution is not a simple  $q$  function, but instead a resource allocation *policy* (a mapping from states to actions) which controls the process. Predictive RA has been investigated extensively over the past decades. In this paper we focus on proactive and reactive solutions, only.

### 3. Distributed Resource Allocation

In this section a few widespread distributed resource allocation approaches will be overviewed and their key properties, such as the guarantees of finding an optimal (or a near optimal) solution, their robustness against different disturbances, such as breakdowns, or against imprecise, uncertain models, will be investigated, with a special emphasis on their adaptive capabilities.

A multi-agent system is a special distributed system with localized decision-making and, usually, localized storage. An agent is basically a self-directed (mostly software) entity with its own value system and a means to communicate with other such objects (Baker 1998). For a general survey on the application of multi-agent systems in manufacturing, see (Monostori, *et al.*, 2006).

#### 3.1 The PROSA Architecture

A basic agent-based architecture for manufacturing systems is PROSA (Van Brussel, *et al.*, 1998). The general idea underlying this approach is to consider both the resources (e.g., machines) and the jobs (interconnected tasks) as active entities. The standard architecture of the PROSA approach (see Figure 1) consists of three types of basic agents: order agents (internal logistics), product agents (process plans), and resource agents (resource handling). However, the PROSA architecture in itself is only a general framework, it does not offer any direct resource allocation solutions.

PROSA is a starting point for the design and development of multi-agent manufacturing control. Resource agents correspond to physical parts (production resources in the system, such as factories, shops, machines, furnaces, conveyors, pipelines, material storages, personnel, etc.), and contain information processing part that controls the resource. Product agents hold the process and product knowledge to assure the correct making of the product. They act like information server to other agents. Order agents represent a

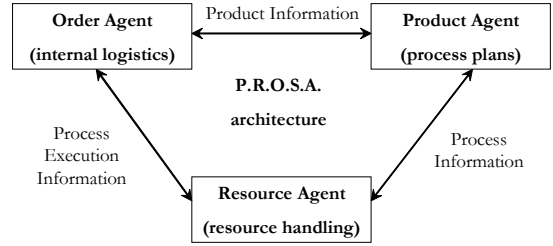


Fig. 1 The PROSA reference architecture.

task or a job (an ordered set of tasks) in the manufacturing system. They are responsible for performing the assigned work correctly, effectively and on time.

#### 3.2 Swarm Optimization

A lot of distributed optimization techniques were inspired by various biological systems (Kennedy and Eberhart, 1995), such as bird flocks, wolf packs, fish schools, termite hills or ant-colonies. These approaches can show up strongly robust and parallel behavior.

The ant-colony optimization algorithm (Moyson and Manderick, 1988) is, in general, a randomized algorithm to solve *Shortest Path (SP)* problems in graphs. It can be shown that RAs can be formalized as SP problems.

The PROSA architecture can also be extended by ant-colony type optimization methods (Hadeli, *et al.*, 2004), in that case a new type of agent is introduced, called as ant. Agents of this type are mobile and they gather and distribute information in the manufacturing system. Their main assumption is that the agents are much faster than the ironware that they control, and that makes the system capable to forecast. Agents are faster and therefore can emulate the system's behaviour several times before the actual decision is taken. The resource allocation in this system is made by local decisions. Each order agent sends ants (mobile-agents), which are moving downstream in a virtual manner. They gather information about the possible schedules from the resource agents and than they return to the order agent with the information. The order agent chooses a schedule and than it sends ants to book the needed resources. After that the order agent regularly sends booking ants to re-book the previously found best schedule, because if the booking is not refreshed then it evaporates (like the pheromone in the analogy of food-foraging ants) after a while. From time to time the order agent sends ants to survey the possible new (and better) schedules. If they find a better solution, the order agent sends ants to book the resources that are needed for the new schedule and the old booking information will simply evaporate.

Swarm optimization methods are very robust, they can naturally adapt to environmental changes, since the ants continuously explore the current situation and the obsolete data simply evaporates if not refreshed regularly. However, these techniques often have the disad-

vantage that finding an optimal or even a relatively good solution cannot be easily guaranteed, theoretically. For example, the ant-colony based extension of PROSA faces almost exclusively the routing problem in resource allocation and it mostly ignores sequencing problems, namely, the efficient ordering of the tasks.

### 3.3 Negotiation-Based Approaches

There are multi-agent systems which use some kinds of negotiation or market-based mechanism (Márkus, *et al.*, 1996). In this case, the tasks or the jobs are associated with order agents, while the resources are controlled by resource agents, like in the case of PROSA.

Market-based resource allocation is a recursive, iterative process with announce-bid-award cycles. During RA the tasks are announced to the agents that control the resources, and they can bid for the available works. The jobs or tasks are, usually, announced one by one, which can lead to myopic behavior and, therefore, guaranteeing an optimal or even an approximately good solution is often very hard.

Regarding adaptive behavior, market-based RA is often less robust than in the case of swarm, e.g., ant-colony based, optimization methods.

### 3.4 Distributed Constraint-Satisfaction

Resource allocation problems (at least their deterministic variants) can be often formulated as constraint-satisfaction problems (Modi, *et al.*, 2001). In this case, they aim at solving the following problem:

$$\begin{aligned} & \text{optimize} && f(x_1, x_2, \dots, x_n), \\ & \text{subject to} && g_j(x_1, x_2, \dots, x_n) \leq c_j, \end{aligned}$$

where  $x_i \in \Omega_i$ ,  $i \in \{1, \dots, n\}$  and  $j \in \{1, \dots, m\}$ . Functions  $f$  and  $g_j$  are real-valued and  $c_j \in \mathbb{R}$ , as well. Most RA problems, e.g., resource constrained project scheduling, can be formulated as a linear programming problem, which formulation can be written as

$$\begin{aligned} & \text{optimize} && \langle c, x \rangle, \\ & \text{subject to} && Ax \leq b, \end{aligned}$$

where  $A \in \mathbb{R}^{n \times m}$ ,  $c \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$  and  $\langle \cdot, \cdot \rangle$  denotes inner product. Then, distributed variants of constrained optimization approaches can be used to compute a solution. In that case, a close-to-optimal solution is often guaranteed, however, the computation time is usually large. The main problems with these approaches are that they cannot take uncertainties into account and, moreover, they are not robust against disturbances.

### 3.5 Problem Decomposition

The idea of divide-and-conquer is often applied to decrease computational complexity in combinatorial optimization problems. The main idea is to decompose the

problem and solve the resulted sub-problems independently. In most cases calculating the sub-solutions can be done in a distributed way (Wu and Zhang, 2005).

These approaches can be effectively applied in many cases, however, defining a decomposition which guarantees both efficient computational speedup together with the property that combining the optimal solutions of the sub-problems results in a global optimal solution is very demanding. Therefore, when we apply decomposition, we usually have to give-up optimality and satisfy with fast but sometimes far-from-optimal solutions. Moreover, it is hard to make these systems robust against disturbances. Tracking environmental changes can be often accomplished by the complete recalculation of the whole solution, only.

## 4. Machine Learning and Resource Control

Machine learning techniques represent a promising new way to deal with resource allocation problems in complex, uncertain and changing environments. These problems can be often formulated as Markov decision processes and they can be solved by *Reinforcement Learning (RL)* algorithms (Zhang and Dietterich, 1995, Ueda, *et al.*, 2000, Aydin and Öztemel, 2000, Csáji, *et al.*, 2003, Csáji and Monostori, 2006).

Now, we propose an RL based adaptive sampler to compute an approximately optimal resource control policy in a distributed way. The sampling is done by iteratively simulating the resource control process. After each trial the policy is refined through recursive updates on the value function using the actual result of the simulation. Thus, from an abstract point of view, the optimization is accomplished through adaptive sampling in the search space. To achieve this, the RAP must be reformulated as controlled Markov process.

### 4.1 Markov Decision Processes

Sequential decision making under uncertainty is often modeled by MDPs. This section contains the basic definitions and some preliminaries. By a (finite, discrete-time, stationary, fully observable) *Markov Decision Process* (MDP) we mean a stochastic system that can be characterized by an 8-tuple  $\langle \mathbb{X}, \mathbb{T}, \mathbb{A}, \mathcal{A}, p, g, \alpha, \beta \rangle$ , where the components are:  $\mathbb{X}$  is a finite set of discrete states,  $\mathbb{T} \subseteq \mathbb{X}$  is a set of *terminal states*,  $\mathbb{A}$  is a finite set of control *actions*.  $\mathcal{A} : \mathbb{X} \rightarrow \mathcal{P}(\mathbb{A})$  is the *availability function* that renders each state a set of actions available in that state where  $\mathcal{P}$  denotes the power set. The *transition function* is given by  $p : \mathbb{X} \times \mathbb{A} \rightarrow \Delta(\mathbb{X})$  where  $\Delta(\mathbb{X})$  is the space of probability distributions over  $\mathbb{X}$ . Let us denote by  $p(y|x, a)$  the probability of arrival at state  $y$  after executing action  $a \in \mathcal{A}(x)$  in state  $x$ . The *immediate cost function* is defined by  $g : \mathbb{X} \times \mathbb{A} \times \mathbb{X} \rightarrow \mathbb{R}$ , where  $g(x, a, y)$  is the cost of arrival at state  $y$  after taking action  $a \in \mathcal{A}(x)$  in state  $x$ . We consider discounted MDPs and the *discount rate* is denoted by  $\alpha \in [0, 1)$ . Finally,  $\beta \in \Delta(\mathbb{X})$  determines the *initial probability distribution* of the states in the stochastic system.

A (stationary, randomized, Markov) control *policy* is a function from states to probability distributions over actions,  $\pi : \mathbb{X} \rightarrow \Delta(\mathbb{A})$ . The initial probability distribution  $\beta$ , the transition probabilities  $p$  together with a control policy  $\pi$  completely determine the progress of the system in a stochastic sense, namely, it defines a homogeneous Markov chain on  $\mathbb{X}$ .

The *cost-to-go* function of a control policy is  $Q^\pi : \mathbb{X} \times \mathbb{A} \rightarrow \mathbb{R}$ , where  $Q^\pi(x, a)$  gives the expected cumulative [discounted] costs when the system is in state  $x$ , it takes control action  $a$  and it follows policy  $\pi$  thereafter

$$Q^\pi(x, a) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \alpha^t G_t^\pi \mid X_0 = x, A_0 = a \right], \quad (1)$$

where  $G_t^\pi = g(X_t, A_t^\pi, X_{t+1})$ ,  $A_t^\pi$  is selected according to policy  $\pi$  and  $X_{t+1}$  has  $p(X_t, A_t^\pi)$  distribution.

A policy  $\pi_1 \leq \pi_2$  if and only if  $\forall x \in \mathbb{X}, \forall a \in \mathbb{A} : Q^{\pi_1}(x, a) \leq Q^{\pi_2}(x, a)$ . A policy is called *optimal* if it is better than or equal to all other control policies. The objective in MDPs is to compute a near-optimal policy.

There always exists at least one optimal (even stationary and deterministic) control policy. Although, there may be many optimal policies, they all share the same unique optimal action-value function, denoted by  $Q^*$ . This function must satisfy a (Hamilton-Jacoby-) *Bellman type optimality equation* (Bertsekas, 2001):

$$Q^*(x, a) = \mathbb{E} \left[ g(x, a, Y) + \alpha \min_{B \in \mathcal{A}(Y)} Q^*(Y, B) \right], \quad (2)$$

where  $Y$  is a random variable with  $p(x, a)$  distribution.

From an action-value function it is straightforward to get a policy, for example, by selecting in each state in a greedy way an action producing minimal costs.

## 4.2 Adaptive Sampling

General RAPs with stochastic durations can be formulated as MDPs, as shown in (Csáji and Monostori, 2006). Then, the challenge of finding a good policy can be accomplished by approximate Q-learning. In that case, the possible occurrences of the resource control process is iteratively simulated, starting from the initial stage of the resources. Each trial produces a sample trajectory that can be described as a sequence of state-action pairs. After each trial, the approximated values of the visited pairs are updated by the Q-learning rule.

The one-step Q-learning rule is  $Q_{t+1} = TQ_t$ , where

$$(TQ_t)(x, a) = (1 - \gamma_t(x, a)) Q_t(x, a) + \gamma_t(x, a) \left[ g(x, a, y) + \alpha \min_{b \in \mathcal{A}(y)} Q_t(y, b) \right], \quad (3)$$

where  $y$  and  $g(x, a, y)$  are generated from the pair  $(x, a)$  by simulation, that is, according to distribution  $p(x, a)$ ; the coefficients  $\gamma_t(x, a)$  are called the *learning rate* and  $\gamma_t(x, a) \neq 0$  only if  $(x, a)$  was visited during trial  $t$ . It is known well (Bertsekas 2001) that if for all  $x$  and

$a: \sum_{t=1}^{\infty} \gamma_t(x, a) = \infty$  and  $\sum_{t=1}^{\infty} \gamma_t^2(x, a) < \infty$ , the Q-learning algorithm will converge with probability one to the optimal value function in the case of lookup table representation. Because the problem is acyclic, it is advised to apply *prioritized sweeping*, and perform the backups in an opposite order in which they appeared during simulation, starting from a terminal state.

To balance between *exploration* and *exploitation*, and so to ensure the convergence of Q-learning, we can use the standard Boltzmann formula (Bertsekas 2001).

## 4.3 Cost-to-Go Approximation

In systems with large state spaces, the action-value function is usually approximated by a (typically parametric) function. Let us denote the space of action-value functions over  $\mathbb{X} \times \mathbb{A}$  by  $\mathcal{Q}(\mathbb{X} \times \mathbb{A})$ . The method of fitted Q-learning arises when after each trial the action-value function is projected onto a suitable function space  $\mathcal{F}$  with a possible error  $\epsilon > 0$ . The update rule becomes  $Q_{t+1} = \Phi T Q_t$ , where  $\Phi$  denotes a projection operator to function space  $\mathcal{F}$ . In (Csáji and Monostori, 2006) support vector regression is suggested to effectively maintain the cost-to-go function. The value estimation then takes the form as follows

$$\tilde{Q}(x, a) = \sum_{i=1}^l (w_i^* - w_i) K(y_i, y) + b, \quad (4)$$

where  $K$  is an inner product kernel,  $y = \phi(x, a)$  represents some peculiar features of  $x$  and  $a$ ,  $w_i, w_i^*$  are the weights of the regression and  $b$  is a bias. As a kernel the usual choice is a Gaussian type function  $K(y_1, y_2) = \exp(-\|y_1 - y_2\|^2 / \sigma^2)$  where  $\sigma > 0$ .

Partitioning the search space by decomposing the problem and applying limited-lookahead rollout algorithms in the initial stage can also speed up the computation considerably (Csáji and Monostori, 2006).

## 4.4 Distributed Sampling

In this section we investigate, how the presented sampling can be distributed among several processors, even if the value function is local to each processor.

If a common (global) storage is available to the processors, then it is straightforward to parallelize the sampling-based approximate cost-to-go function computation: each processor can search independently by making trials, however, they all share (read and write) the same global cost-to-go function. They update the value function estimations asynchronously.

A more complex situation arises when the memory is completely local to the processors, which is realistic if they are physically separated, e.g., in a GRID.

A way of dividing the computation of a good policy among several processors is possible when there is only one "global" value function, however, it is stored in a distributed way. Each processor stores a part of the value function and it asks for estimations which it

requires but does not have from the others. The applicability of this approach lies in the fact that the underlying MDP is acyclic and, thus, it can be effectively partitioned, for example, by starting the trials of each processor from a different starting state.

If the processors have their own completely local value functions, they could have widely different estimations on the optimal state-action values. In order to effectively compute a global value function, the processors should count how many times did they update the estimations of the different pairs. Finally, the values of the global Q-function can be combined from the individual estimations by a Boltzmann formula.

## 5. Experimental Results

The proposed RL based approach was tested on Hurink’s benchmark dataset (Hurink, *et al.*, 1994). It contains *Flexible Job-Shop (FJS)* scheduling problems with 6–30 jobs (30–225 tasks) and 5–15 resources. The performance measure is make-span, thus, the total completion time has to be minimized. These problems are “hard”, which means, e.g., that standard dispatching rules or heuristics perform poorly on them. This dataset consists of four subsets, each subset containing about 60 problems. The subsets (sdata, edata, rdata, vdata) differ on the ratio of resource interchangeability, shown in the “parallel” column in the table (Figure 2). The columns with label “x iter.” show the average error after carrying out “x” iterations. The execution of 10000 simulated trials (after on the average the system has achieved a solution with less than 5% error) takes only few seconds on a common computer of today.

dataset	parallel	1000 iter.	5000 iter.	10000 iter.
sdata	1	8.54 %	5.69 %	3.57 %
edata	1.2	12.37 %	8.03 %	5.26 %
rdata	2	16.14 %	11.41 %	7.14 %
vdata	5	10.18 %	7.73 %	3.49 %
<b>average</b>	<b>2.3</b>	<b>11.81 %</b>	<b>8.21 %</b>	<b>4.86 %</b>

**Fig. 2** Benchmark dataset of FJS problems.

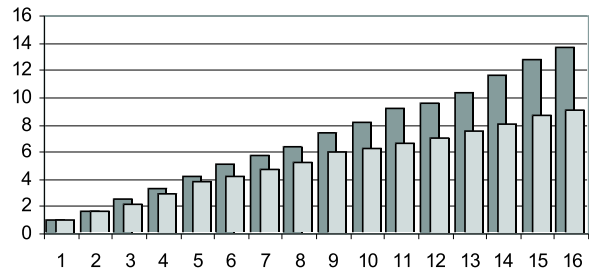
We initiated experiments on a simulated factory by modeling the structure of a real plant producing customized mass-products. We have used randomly generated orders (jobs) with random due dates. The tasks and the process-plans of the jobs, however, covered real products. In this plant the machines require product-type dependent setup times, and another specialty of the plant is that, at some previously given time points, preemptions are allowed. The applied performance measure was to minimize the number of late jobs and an additional secondary performance measure was to minimize the total cumulative lateness, which can be applied to comparing two situations having the same number of late jobs. In Figure 3 the convergence speed (average error) relative to the number of resources and

tasks is demonstrated. The workload of the resources was approximately 90%. The results show, that our adaptive sampling based resource control algorithm can perform efficiently on large-scale problems.

resources	tasks	1000 iter.	5000 iter.	10000 iter.
16	140	4.26 %	3.28 %	2.45 %
25	280	7.05 %	4.15 %	3.61 %
30	560	7.56 %	5.96 %	4.57 %
50	2000	8.69 %	7.24 %	6.04 %
<b>100</b>	<b>10000</b>	<b>15.07 %</b>	<b>10.31 %</b>	<b>9.11 %</b>

**Fig. 3** Industry related simulation experiments.

We have also investigated the parallelization of the method, namely, the speedup of the system relative to the number of processors. The average number of iterations was studied, until the system could reach a solution with less than 5% error on Hurink’s dataset.



**Fig. 4** Average speedup in case of distributed sampling with global and local value functions.

We have treated the average speed of a single processor as a unit (cf. with the data in Figure 2). In Figure 4 the horizontal axis represents the number of applied processors, while the vertical axis shows the relative speedup achieved. We applied two kinds of parallelization: in the first case (dark gray bars), each processor could access a global value function. It means that all of the processors could read and write the same global action-value function, but otherwise, they searched independently. In that case the speedup was almost linear. In the second case (light gray bars), each processor had its own, completely local action-value function and, after the search was finished, these individual functions were combined. The experiments show that the computation of the RL based resource control can be effectively distributed, even if there is not a commonly accessible action-value function available.

## 6. Concluding Remarks

Efficient allocation of reusable resources over time in uncertain and dynamic environments is an important problem in many real-world domains. The paper overviewed some distributed RA approaches and presented an RL based adaptive solution, as well.

There are several advantages why RL based solutions are preferable to other kinds of distributed approaches described above. These favorable features are:

- (1) RL methods are robust, they essentially face the problem under the presence of uncertainties.
- (2) They can quickly adapt to unexpected changes in the environmental dynamics, such as breakdowns. This property can be explained by the Lipschitz type dependence of the optimal value function on the transition probabilities and the rewards.
- (3) Additionally, there are theoretical guarantees of finding optimal (or approximately optimal) solutions, at least in the limit.
- (4) Moreover, the actual convergence speed is usually high, especially in the case of applying distributed sampling or problem decomposition.
- (5) Additionally, the resulted distributed RL based resource allocation is almost scale-free, it can effectively face large-scale problems without dramatic retrogression in the performance.
- (6) Finally, the proposed method constitutes an any-time solution, since the sampling can be stopped after any number of iterations.

Therefore, RL seems one of the most promising approaches for distributed RA in real-world domains.

### Acknowledgements

This research was partially supported by the NKFP Grant No. 2/010/2004 and by the OTKA Grant No. T049481. Balázs Csanád Csáji greatly acknowledges the scholarship of the Hungarian Academy of Sciences.

### References

- Aydin, M. E., Öztemel, E. (2000). Dynamic Job-Shop Scheduling Using Reinforcement Learning Agents. *Robotics and Autonomous Systems*, Vol. 33, 169–178. Elsevier Science
- Baker, A. D. (1998). A Survey of Factory Control Algorithms That Can Be Implemented in a Multi-Agent Heterarchy: Dispatching, Scheduling, and Pull. *Journal of Manufacturing Systems*, Vol. 17, 297–320. Society of Manufacturing Engineers
- Bertsekas, D. P. (2001). *Dynamic Programming and Optimal Control*. 2nd edition. Athena Scientific
- Csáji, B. Cs., Kádár, B., Monostori, L. (2003). Improving Multi-Agent Based Scheduling by Neurodynamic Programming. In: *Proceedings of the 1st International Conference on Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Artificial Intelligence*, Vol. 2744, pp. 110–123.
- Csáji, B. Cs., Monostori, L. (2006). Adaptive Sampling Based Large-Scale Stochastic Resource Control. In: *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-06)* [in print]
- Dolgov, D. A., Durfee, E. H. (2004). Optimal Resource Allocation and Policy Formulation in Loosely-Coupled Markov Decision Processes. *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pp. 315–324.
- Hadeli, Valckenaers, P., Kollingbaum, M., Van Brussel, H. (2004). Multi-Agent Coordination and Control Using Stigmergy. *Computers in Industry*, Vol. 53, 75–96. Elsevier Science
- Hurink, E., Jurisch, B., Thole, M. (1994). Tabu Search for the Job-Shop Scheduling Problem with Multi-Purpose Machines. *Operations Research Spektrum*, Vol. 15, 205–215.
- Kennedy, J., Eberhart, R. C. (1995). Particle Swarm Optimization. *IEEE International Conference on Neural Networks*, Vol. 4, 1942–1948.
- Márkus, A., Kis, T., Váncza, J., Monostori, L. (1996). A Market Approach to Holonic Manufacturing. *Annals of the CIRP*, Vol. 45, 433–436.
- Modi, P. J., Hyuckchul, J., Tambe, M., Shen, W., Kulkarni, S. (2001). Dynamic Distributed Resource Allocation: Distributed Constraint Satisfaction Approach. *Pre-proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages*, pp. 181–193.
- Monostori, L., Váncza, J., Kumara, S. R. T. (2006). Agent-Based Systems for Manufacturing. *Annals of the CIRP*, Vol. 55, No. 2 [paper sent for review]
- Moyson, F., Manderick, B.: The Collective Behaviour of Ants : an Example of Self-Organization in Massive Parallelism. In: *Proceedings of AAAI Spring Symposium on Parallel Models of Intelligence*, Stanford, California, 1988.
- Perkins, J. R., Humes, C., Kumar, P. R. (1994). Distributed Scheduling of Flexible Manufacturing Systems: Stability and Performance. *IEEE Transactions on Robotics and Automation*, Vol. 10, 133–141. IEEE Press
- Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall
- Ueda, K., Hatono, I., Fujii, N., Vaario, J. (2000). Reinforcement Learning Approaches to Biological Manufacturing Systems. *Annals of the CIRP*, Vol. 49, 343–346.
- Ueda, K., Márkus, A., Monostori, L., Kals, H. J. J., Arai, T. (2001). Emergent Synthesis Methodologies for Manufacturing. *Annals of the CIRP*, Vol. 50, 535–551.
- Van Brussel, H., Jo Wyns, Valckenaers, P., Bongaerts, L., Peeters, P. (1998). Reference Architecture for Holonic Manufacturing Systems: PROSA. *Computers in Industry*, Vol. 37, 255–274.
- Williamson, D. P., Hall L. A., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevastjanov, S. V., Shmoys, D. B. (1997). Short Shop Schedules. *Operations Research*, Vol. 45, 288–294.
- Wu, T., Ye, N., Zhang, D. (2005). Comparison of Distributed Methods for Resource Allocation. *International Journal of Production Research*, Vol. 43, 515–536. Taylor and Francis
- Zhang, W., Dietterich, T. (1995). A Reinforcement Learning Approach to Job-Shop Scheduling. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1114–1120.