A Reinforcement Learning Approach for Supply Chain Management

Tim Stockheim, Michael Schwind, and Wolfgang Koenig

Chair of Economics, esp. Information Systems,
Frankfurt University, D-60054 Frankfurt, Germany,
stockheim,schwind,koenig@wiwi.uni-frankfurt.de,
WWW home page: http://www.wi-frankfurt.de

Abstract. The paper presents a decentralized supply chain management approach based on reinforcement learning. Our supply chain scenario consists of loosely coupled yield optimizing scheduling agents trying to learn an optimal acceptance strategy for the offered jobs. The optimizer constructs a mandatory schedule by inserting the requested jobs, which arrive stochastically from the customers, gradually into a production queue if the job yields a sufficient return. To reduce complexity the agents are divided into three components. A supply chain interface, classifying job offers, a *reinforcement learning algorithm* component, which makes the acceptance decision and a *deterministic scheduling component*, which processes the jobs and generates a preliminary state space compression. The reinforcement learning algorithm accepts offers according to their delivery due date, the job price, the timeout penalty cost, and the information provided by the scheduling component. The tasks are finally executed on the suppliers machine following the queue's schedule. In a performance comparison of our yield optimizing agent it turns out that the reinforcement learning solution outperforms the simple acceptance heuristic for all training states.

1 Learning Agents in Decentralized Supply Chain Optimization

Supply chain management (SCM) is concerned with the efficient allocation of production resources in a cross plant environment. Besides the allocation aspect, profit considerations of the individual supply chain partners / competitors play an important role for the attribution of tasks. It is therefore advisable form the economic point of view to employ the distributed decision structures of *multi agent systems* for the SCM task optimization and allocation process (Barbuceanu & Fox 1996) (Walsh & Wellman 1999). Taking the dependencies of the underlying production techniques into account, the SCM allocation problem presents itself as an algorithmic problem of combinatorial complexity (NP-hard). One way to deal with this problem is to create time dependent price functions and to employ *stochastic optimization methods* to attain a near optimal allocation (Stockheim, Schwind, Wendt & Grolik 2002). Other approaches address the combinatorial allocation problem aspect directly while using auctions, especially the *combinatorial auction*, which is able to take account of the utility interdependencies of the negotiated goods and resource bundles (Walsh, Wellman, Wurman & MacKie-Mason 1998).

Pontrandolfo, Gosavi, Okogbaa & Das (2002) provide a *global supply chain management* (GSCM) approach based on a *reinforcement learning algorithm* (RLA) called SMART (Semi Markov Average Reward Technique). The GSCM component has to learn the optimal procurement and distribution policy for a supply system consisting of production plants, export warehouses, import warehouses and final markets in three countries. While calculating

production and transportation costs using currency exchange rates, tariffs, production-, inventory, late delivery and transportation costs, the RLA chooses between three possible suppliers and one of two transportation modes. SMART is tested by applying various demand patterns to the supply chain, grounding on a Erlang probability distribution modified by diverse mean and deviation parameters. Compared with two heuristics, one called LH standing for *local heuristic* preferring a inner country production and distribution policy and another denoted as BH *balanced heuristic* while issuing demand mainly to warehouses with low capacity load, the SMART allocation mechanism provides the highest reward.

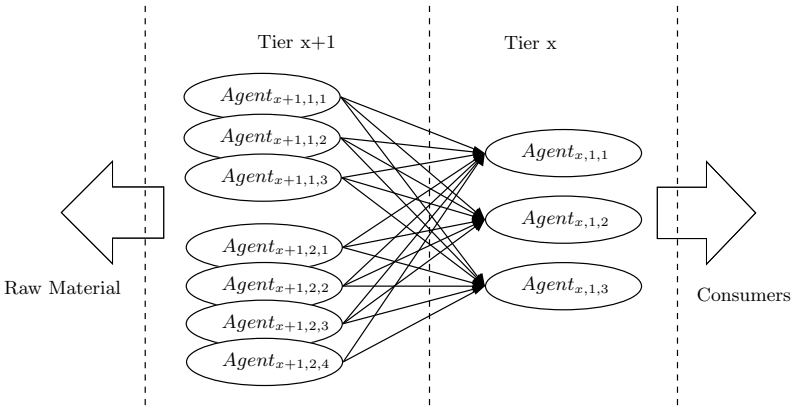

Fig. 1. Decentralized SCM-network employing RL-agents

In our decentralized supply chain management (DSCM) scenario we employ a network of yield optimizing scheduling (YOS) agents (Fig. 1) that act on their own behalf, each of them organizing a machine capable to manufacture a single supply product needed to to complete the final product of the SCM web. The agents are responsible for the machines load schedule, while trying to optimize the return yielded by the completed manufacturing tasks. If a supplier agent gets a request for the delivery of a product from a customer agent[1] it checks the filling degree of its production scheduling queue and decides about the acceptance or rejection of the production request according to the job price, delivery due date, accomplishment priorities and the expectation value for the future arrival of tasks yielding a higher total income. Each supply product, required in the next manufacturing tier has to be provided by at least three other agents to guarantee a market situation that enables the agents independent acceptance decisions, without putting the operation of the supply chain at risk. The source of demand is the customer coupled with the third tier SCM agents by typical marketing related price functions.[2] Triggered by the consumer demand third tier agents generate production task requests for the second tier agents.[3] The requests can be accepted or rejected by each of the agents, responsible for the production of the required supply parts. The customer agent contracts with the supplier agents accepting the

[1] As can be seen in Fig. 1 RL-SCM-network agents act as supplier to the lower tier side and as customer in direction of the higher tier.

[2] This coupling is not in the scope of our research, nevertheless it is important to emphasize that the third tier to consumer price mechanism differs strongly from intra-SCM mechanisms discussed here.

[3] Material flow is directly mapped to production capacity requirements given by the supply product ratio of the technical part lists in our model.

requests until the procurement of the needed product quantity is guaranteed. If there is no sufficient supply the customer agent has to increase the bid price for the needed items. This decentralized procurement process is extended to the remaining tiers of the supply chain. Due to the agents yield and capacity maximizing learning mechanism the SCM system reaches a stable state, that establishes a market equilibrium price for each supply product depending on demand and resource capacity availability.

2 Reinforcement Learning

The reinforcement learning approach generally grounds on the idea to enable (software-) agents to explore a finite state space S by using *trial&error* while recording the experience (Sutton & Barto 1998). During learning the agent receives rewards r or penalties p while reaching key positions and obstacles (states s) in S. Repeating this exploration continuously, the agent records the received rewards r for the previous state constituting knowledge about distant targets and obstacles. After an adequate learning phase the agent knows the shortest path to the highest reward position $s(r^*)$. Considering the base elements of *reinforcement learning* - states s, actions a and rewards r - under the MAS aspect a system model depicted in Fig. 2 could be assumed (Sutton & Barto 1998).

- The reinforcement learning agent (RL-agent) is connected to his environment via sensors.
- In every step of interaction the agent receives a feedback about the state of the environment s_{t+1} and the reward r_{t+1} of its latest action a_t.
- The agent chooses an action a_{t+1} representing the output function, which changes the state s_{t+2} of environment.
- The agent anew gets a feedback, through the reinforcement signal r_{t+2}.
- Objective of the agent is the maximization of the aggregated reinforcement signals.

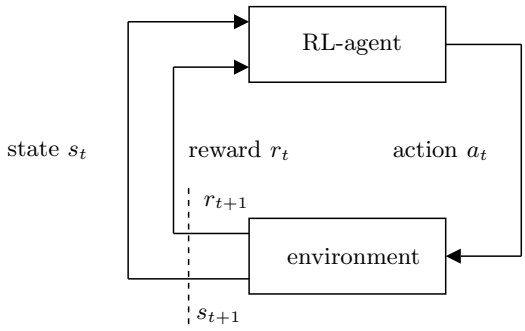

Fig. 2. RL-agent in environment

To determine the optimal path of actions a_t in a decision optimization environment it is normally necessary to evaluate all states s_t. The use of a myopic decision strategy while selecting the next action with the highest reward r_t does not guarantee to obtain the optimal decision path, because a firstly favorable branch from state s_{t+1} to s_{t+2} could turn out as suboptimal later on. Bellman (1957) solved this problem by recursively calculating the optimal decisions in each state s_t while introducing *stochastic dynamic programming* (SDP).

Calculation of the optimal path is done by iteratively approximating the state values employing the *Bellman* equation 1. The state value $V^{\pi}(s_t)$ equation 1 of a policy π is defined as the expected value of discounted state value $V^{\pi}(s_{t+1})$ of the next state s_{t+1} summed up with the respective expected reward r_{t+1}. The usage of the discount factor γ in equation 3 is not essential to SDP but turns out to be valuable in terms of convergence behavior for reinforcement learning. In addition to the state value the action value $Q^{\pi}(s_t, a_t)$ of a policy π is defined. As can be seen from equation 2 it depends on the expectation value of a specific action taken in s_t.

$$V^{\pi}(s_t) \;=\; E_{\pi}\left\{\, R_t |\, s_t \,\right\} \;=\; E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1}\right\} \tag{1}$$

$$Q^{\pi}(s_t, a_t) \;=\; E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma_{t+k+1}^{r}\right\} \tag{2}$$

To maximize the sum of rewards it is necessary to calculate the optimal state value and the optimal action value respectively equation 3 and 4:

$$V^{*}\left(s_t\right) \;=\; \max_{\pi} V^{\pi}(s_t) \tag{3}$$

$$Q^{*}\left(s, a\right) \;=\; E\left\{r_{t+1} + \gamma V^{*}(s_{t+1})\right\} \tag{4}$$

An alternative way of writing equation 1 is given in equation 5. The computation of $V^{*}(s_t)$ is done by repeatedly calculating state values $V_{k+1}(s_t)$. This procedure is known as value iteration.

$$V_{k+1}(s_t) \;=\; \max_{a} E\left\{r_{t+1} \;+\; \gamma \sum_{s_{t+1}} V_k(s_{t+1})\right\} \tag{5}$$

Alternatively a policy adjustment during the evaluation could be applied to avoid the necessity for exploring all decisions to rate a policy. Such technique is called policy iteration because only decision policies are changed. While using policy iteration it must be ascertained that a change from decision policy π to policy π' in state s_t leads to a refinement of $V^{\pi}(s_t)$. A change in policy π can be evaluated in terms of an improvement for all potential actions a_t in state s_t, as far as the value function $V^{\pi}(s_t)$ of policy π is known in s_t. Selecting the optimal policy in every visited state s_t while regarding only the next step leads to a greedy strategy for permanent policy improvement. After optimizing a decision policy π yielding the improved policy π' by the usage of $V^{\pi}(s_t)$, a new value function $V^{\pi'}(s_t)$ can be calculated to improve π' to π'' etc. Policy iteration implies an alternation between of improvement and evaluation toward a greedy strategy.
Whereas the usage of SDP requires the determination of $V(s)$ for all feasible system states, the application of a *Monte-Carlo method*(MCM) only demands the estimation of a partial set of the $V(s)$. It is therefore sufficient to calculate the $V(s)$ of several traces (episodes). While passing through the episodes the mean value of the rewards is used as state value estimator. Two methods are common:

- *First-visit MC-method*:
  The episodes are passed through, while recording the mean of the previous rewards in each visited node. The fist recorded value is preserved even if a state is visited multiple times.
- *Every Visit MC-method*:
  In contrast to the first-visit MC-method, the every-visit method employs an update rule for $V(s)$, e.g. the constant-α MC-method:

$$V^{\pi}_{new}(s_t) = V(s_t) + \alpha\,[R_t - V(s_t)] \tag{6}$$

A randomized strategy is crucial for a satisfying estimation of the state values in policy iteration. Starting with an arbitrary policy an episode is generated in the evaluation phase. Along the episodes trace action values are generated using the first- or every-visit method. The improvement phase follows yielding an optimization of the selected policy by adjusting the decision making process to the maximum action value. To enable a learning effect the MC-algorithm provides a stochastic element making exploration possible. This done while selecting the path with the optimal action value $Q(s,a)$ only with probability $1-\varepsilon$. In case of this ε-greedy policy traces with an initially poorer estimator for the action value can be chosen making exploration of new episodes with higher long term return possible.
While SDP enables the selection of an optimal action a for the next state s on a decision level k, as far as all actions have been evaluated the MCM depends on the evaluation of at least one complete episode containing state s to calculate the optimal Q-values. *Temporal difference learning* (TDL) avoids the disadvantages of both methods. Beginning with the constant-α MCM the horizon of an episode can be shortened to one decision step guaranteeing convergence for sufficient small α:

$$V^{\pi}(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{7}$$

Based on equation 7 the TDL-algorithm Tab. 1 can be formulated.

```
Initialize V(s_t) and evaluation policy π arbitrary
Repeat for each episode
    Initialize s_t
    Repeat (for each step of the episode):
        Perform action a_t, observe reward r_{t+1}
        and the next state s_{t+1}
        V(s_t) ← V(s_t) + α [r_{t+1} + γV(s_{t+1}) − V(s_t)]
        s_t ← s_{t+1}
    until s_t is terminal state
```

Table 1. Temporal Difference Learning Algorithm

TDL-methods are using estimated values, with regard to two aspects. On the one hand the expected value of $V^{\pi}(s)$ is calculated employing sample-backups (episodes) on the other hand an estimator for the actual state-value V_t replaces the exact V^{π}. TDL employs the policy evaluation and improvement alternation of policy iteration.
A variant of TDL called Q-learning, can be considered as the most important RL-method and has been formulated by Watkins (1989). Q-learning calculates the approximation of the optimal action value function by using only a one period difference.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_{a_t} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \tag{8}$$

The algorithm described here uses the ε-greedy policy for estimation:

3 Scheduling Approaches using Reinforcement Learning

As mentioned in section 1 computational complexity is high for combinatorial optimization problems (COP) like scheduling in supply chain management environments, due to the

```
Initialize Q(s_t, a_t) arbitrary
Repeat for each episode
    Initialize s_t
    Repeat (for each step of the episode):
        Select a_t from s_t using a policy derived from Q
        (e. g. ε-greedy)
        Perform action a_t and observe r_{t+1}, s_{t+1}
        Q(s_t, a_t) ← Q(s_t, a_t) + α [r_{t+1} + γ max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) − Q(s_t, a_t)]
        s_t ← s_{t+1}
    until s_t is terminal state
```

Table 2. Q-Learning Algorithm

complexity of S. Trying to fit priced job requests with precedence constraints onto a yield- and time minimizing schedule represents an instance of a COP.
In the reinforcement learning (RL) literature, dealing with scheduling applications, two main approaches able to handle the computational complexity of the COP, can be identified:

- RL *controls the optimal policy of local search methods*, but does not try to solve the COP itself.
- RL *is applied as a distributed optimization technique*, operating only on the local known states of the individual RL scheduling agent, hoping that emergent system behavior will generate feasible results.

In a first representative work addressing optimal policy search in connection with RL scheduling problems Zhang & Dietterich (1995) use a temporal difference $TD(\lambda)$ algorithm combined with an *artificial neural network* (ANN) to learn a heuristic evaluation function. Starting with a critical-path schedule the algorithm incrementally repairs resource constraint violations using a *reassign operator*, which changes the tasks resource assignment for one of the required resources, and a *move operator*, which shifts a task to a different time slot and reschedules the remaining affected tasks according to the temporal dependencies by using a critical path method. In the original version of this algorithm by Zweben, Daun & Deal (1994), this is done by employing a simulated annealing (SA) approach called (*iterative repair*) IR-algorithm. The RL variant of the IR-algorithm uses the *reassign* and *move* operator as feasible actions as well. Zhang & Dietterich (1995) introduce a *resource dilatation factor* (RDF) to indicate the total over allocation of the resources required for an actual schedule. Goal of the learning is to select the RL actions such, that they reduce the overload as quick as possible. For state space representation the ANN has eight input factors, e.g. the RDF, the percentage of time units violation in the actual schedule etc. In an evaluation of the RL scheduling Zhang & Dietterich (2000) demonstrate that the RL algorithm performs the IR-algorithm out by far.
Boyan & Moore (1998) present a RL system called STAGE, that predicts the outcome of a *local search algorithm*. The system uses the value function $V^{\pi}(s)$ to learn the optimal starting point of an *hill climbing algorithm*. The states passed by a hill climbing episode (trajectory) are updated with the optimization reward r, which is the local optimal value of the objective function, if the optimization result is better then the outcome achieved in previous episodes. State space compression is done using a *polynomial regression network* (PRN). Boyan & Moore (2000) apply STAGE to *global optimization domains* like e.g. *bin packing problems*, *Bayes network structure finding* or *channel routing* and find that it shows a better average performance than traditional SA and hill climbing algorithms. As Chang (1998) points out, this approach is suitable for scheduling problems as well, due to its relation to the COP.

In a further study Schneider, Boyan & Moore (1998) propose an approach constructed similar to the Schwind & Wendt (2002) reinforcement learning yield management system. They train the optimal schedule of a eight week production scenario. Hereby production configuration may be changed just in two week intervals and only 17 schedule configurations are feasible. State space compression is done by alternatively employing 1-nearest neighbor, locally weighted linear regression and global quadratic regression methods. The performance of the system turns out to be better, than that achieved with simulated annealing and greedy optimization methods.
An example for the second mentioned complexity reduction approach in the RL scheduling domain is presented by Riedmiller & Riedmiller (1999). In their work an ANN based agent autonomously optimizes its local task dispatching policy with respect to a global defined optimization goal.

4 Systems Infrastructure

Each YOS agent of the DSCM environment has an identical system infrastructure consisting of three main components (Fig. 3). The first component is the SCM interface, which acts in the context of the YOS agent as *job generator* simulating job offer sequences for test purposes or passes offers submitted by the next tier suppliers to RLA and DSC respectively. Furthermore the *reinforcement learning algorithm* (RLA), which has to learn the yield optimizing job acceptance strategy. As third module the *deterministic scheduling component* (DSC) arranges the accepted jobs in the production queue to avoiding penalty costs caused by not-in-time delivery. Orders resulting from the accepted offers are submitted to the supplier agents. Although the functions of the DSC include procurement it is of no relevance to the simulations presented here. In the next sections we detail the fundamental functions of the YOS agents.

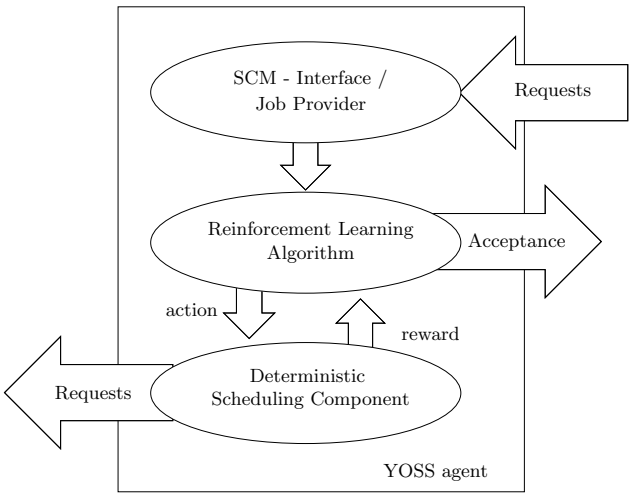

Fig. 3. High level use case diagram

4.1 Properties of Job Requests

The requests for a job j offered to a supplier generated or passed by the JGC and processed by the RLA and DSC have the following attributes,

- o_j which is the offer date of the job (release time of the manufacturing request),
- d_j which is the jobs due date (last date for delivery without necessity of a penalty payment),
- l_j denoting the job length (assumed processing time),
- v_j indicating the value class of a job (class of returns for the fulfilled request) and
- p_j describing the contract penalty class of the job (class of penalty payments for delivery after due date).

4.2 Reinforcement Learning Algorithm

Before designing the functionality of the reinforcement learning algorithm the state space, the action space and the state transition conditions have to be specified. The **state space** S of the RLA consists of three independent parameters,

- $g_f \in G$ representing the schedule queue filling,
- $v_j \in V$ denoting the value class of current offer for job j and
- $p_j \in P$ characterizing the contract penalty class of current job j.

Consequently, the resulting state space amounts to

$$S = G \times V \times P \tag{9}$$

states and is of dimension m

$$m = |G| \cdot |V| \cdot |P| \tag{10}$$

Employing a moderate selection of parameter variables, the state space dimension is low enough to avoid the necessity of an ANN for state space compression in our YOS agent.[4] The **action space** is restricted to the *accept* and *reject* decision for the current job request.

$$A = \{accept, reject\} \tag{11}$$

A **state transition** $\delta(s_t, a_t); s_t \in S, a_t \in A$ is subdivided in several phases, whereof only the second can be affected by the RLA (see Fig. 4). The transition phases consist of

- δ_1: a scheduling phase for the production plan by the DSC and a classification of the requested job coupled with the propagation of the respective parameters to the RLA.
- δ_2: a decision phase concerning the acceptance or rejection of the requested job j_{t+1},
- δ_3: and a notification of the reward r_{t+1} coupled with the update of the RL-system.

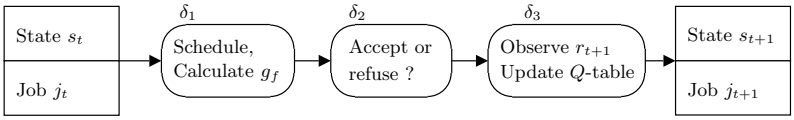

Fig. 4. State transition in detail

[4] Later versions of the YOS agent of course will make use of a state dimension reduction system like an ANN, which is necessary to employ YM-RL scheduling for realistic problem dimensions.

As a result of the previous state space design and the state transitions actions space $Q = \{S \times A\}$ is defined and a description of the RL-YM algorithm can be given. The algorithm grounds on the standard Q-learning depicted in Tab. 2. The characteristic feature of the RL-algorithm designed here, grounds in the fact that the reward is generated by the DSC's scheduling heuristic (line 7 to 8 of Tab. 3). The RLA learns therefore a yield optimal acceptance strategy based on the capability of the scheduling heuristic.

```
Initialize Q(s_t, a_t) arbitrary
Repeat (for a fixed number of episodes):
    Repeat for all jobs in job list of JGC
        Receive a request for job j from JGC at time o_j
        Determine actual state s_t from p_j, v_j and g_f
        Select an action a_t for state s_t (ε-greedy)
        Perform action a_t and inform DSC about acceptance/rejection of job j
        Observe reward r_t determined by the DSC
        Q(s_t, a_t) ← Q(s_t, a_t) + α · [r_t + γ · max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) − Q(s_t, a_t)]
```

Table 3. Reinforcement Learning Yield Management Algorithm

4.3 Deterministic Scheduler Component

After having outlined the RLA of the YOS agents the DSC is now depicted in detail. The parameter g_f represents the filling degree of the production queue under the assumption, that the offered job is accepted while using the amount of provisional allocated tasks, job length l_j, offer date o_j, and due date d_j for calculation. The scheduler component sorts all jobs under the descending order of their due dates. To validate the resulting schedule/queue we compose job ratios g_x using equation 12.

$$g_x = \frac{\sum_{i=1}^{x} l_i - o_j}{d_x - o_j} \tag{12}$$

These job ratios g_x are used to classify the schedule. Table 4 shows the classification we used in our simulations. Our tests showed that other classifications did not yield significant better results.

	job class g_f				
	0	1	2	3	4
$max_x g_x$	0.0 . . . 0.2	0.2 . . . 0.4	0.4 . . . 0.7	0.7 . . . 1.0	> 1.0

Table 4. Classification of jobs according to the time-to-due-date and due-date ratio g_f

Based on these calculations the RLA decides about acceptance or rejection of the current job request. In case of acceptance the DSC maps the job into the production plan.

5 Benchmark Algorithm for the RL-YM Scheduler

To benchmark the performance of the RL-YM scheduler we employed a simple heuristic, which accepts production requests in dependence of the production queues filling level (see Tab. 5). The benchmark algorithm should establish a job acceptance/rejection equilibrium providing a considerable return from production.

```
Initialize scheduler with j = 0
Repeat for each requested job j
    If g_f ≤ 4 (job fits in schedule)
        accept
    Else
        reject
```

Table 5. Benchmark Algorithm for the RL-YM Scheduler

6 Performance of the RL-YM Scheduler

Using a JAVA implementation of the RL-YM algorithm, a test set consisting of 1000 scheduling requests produced by the JGC based on tasks with equally distributed length and Poisson distributed arrival process was employed to learn a yield optimal acceptance policy. Subsequently the performance of the trained RL-YM system was evaluated using a separate set of 1000 scheduling requests. During the learning phase the JGC passes a job request j taken from the training set to the RLA, which has to decide about job acceptance or rejection. After the RLA has accepted a task, the DSC tries to fit the job into the schedule avoiding the occurrence of penalty costs due to not-in-time delivery. Later on, when the scheduled production has been executed, the RLA gets feedback for the acceptance/rejection decision depending on a positive or negative production return. Such the system learns yield maximizing acceptance decisions by updating the Q-values of the corresponding states with positive or negative rewards. Learning is done in various intensities, including an average number of updating visits per state s, accounting from 10 to 200 epochs and varying the exploration intensity ε (see Fig. 5). It turns out, that the RLA performs best for ε below 0.5 depending on the number of training epochs in terms of minimum and average income. Interestingly maximum income is increased significantly for high ε showing that learning with lower exploration rates is not able to reveal all local yield optima in state space. Unsurprisingly this increase of maximum income occurring with higher ε has to be payed for with lower minimal income generating an increasing volatility of return. Considering all significant output parameters of Fig. 5 the 200 epochs learning case performs best in a range of ε between 0.1 and 0.5, showing high average income at low volatility.
Comparing the RL-YM schedulers results to the income gained with the benchmark algorithm the findings described above are confirmed. As can be seen in Fig. 6 the RL algorithm outperforms the average income of the simple learning strategy in the benchmark heuristic for all training states.

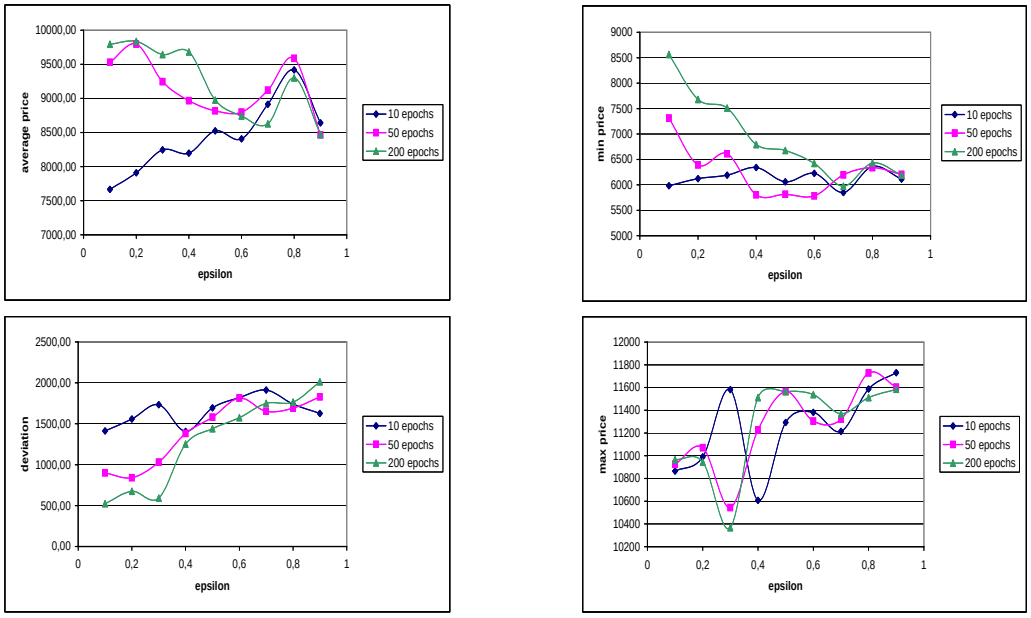

Fig. 5. RL-YM algorithms performance, using a trained system after 10, 50 and 200 epochs of learning with epsilon increasing from 0.1 to 0.9 in steps of 0.1. For each point of the diagram 40 requests series were performed, measuring the minimum, maximum and average return after the simulation series.

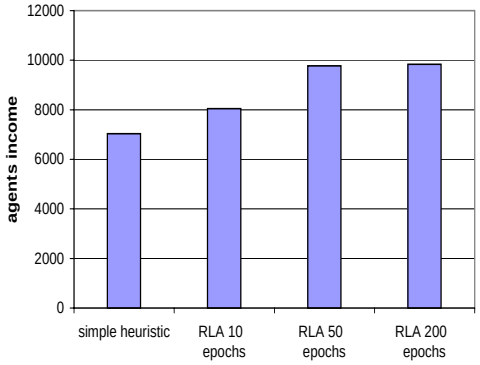

Fig. 6. Average income of RL-simulation runs including 10, 50, 200 learning epochs compared to a simple heuristic

7 Conclusion

By implementing and testing a low parametrized reinforcement learning - yield management scheduling system designed for a multi agent supply chain environment, we could show that the RL-YM solution outperforms a simple learning heuristic in all training states. The system shows satisfying learning performance for ε from 0.1 to 0.5 yielding higher average return providing a low income deviation. Up to now tests were done using only a single scheduling agent, leaving an evaluation of the whole multi agent supply chain system an open issue. Together with the implementation of a state compression mechanism, like an artificial neural network, to enable systems capability for higher state space dimensions, the research of equilibrium building processes in the partial supply markets is the most interesting aspect of the project that should be pursued.

References

Barbuceanu, M. & Fox, M. S. (1996), Coordinating multiple agents in the supply chain, *in* 'Proceedings of the Fifth Workshop on Enabling Technology for Collaborative Enterprises (WET ICE'96), Stanford University, CA', pp. 134–141.

Bellman, R. E. (1957), *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Boyan, J. & Moore, A. (1998), Learning evaluation functions for global optimization and boolean satisfiability, *in* 'Proceedings of the Fifteenth National Conference on Artificial Intelligence', pp. 3–10.

Boyan, J. & Moore, A. (2000), 'Learning evaluation functions to improve optimization by local search', *Journal of Machine Learning Research* **10**, 77–112.

Chang, S. S. (1998), A survey on reinforcement learning in global optimization, Technical report, Computer Science Department of the University of Berkeley, Berkeley, CA.

Pontrandolfo, P., Gosavi, A., Okogbaa, O. G. & Das, T. K. (2002), 'Global supply chain management: a reinforcement learning approach', *International Journal of Production Research* **40**(6), 1266–1317.

Riedmiller, S. C. & Riedmiller, M. A. (1999), A neural reinforcement learning approach to learn local dispatching policies in production scheduling, *in* 'Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'99), Stockholm, Sweden', pp. 764–771.

Schneider, J. G., Boyan, J. A. & Moore, A. W. (1998), Value function based production scheduling, *in* 'Proceedings 15th International Conference on Machine Learning', Morgan Kaufmann, San Francisco, CA, pp. 522–530.

Schwind, M. & Wendt, O. (2002), Dynamic pricing of information products based on reinforcement learning: A yield-management approach, *in* M. Jarke, J. Koehler & G. Lakemeyer, eds, 'KI 2002: Advances in Artificial Intelligence, 25th Annual German Conference on AI (KI 2002), Aachen, Germany', Vol. 2479 of *Lecture Notes in Computer Science*, Springer Verlag, Berlin, Germany, pp. 51–66.

Stockheim, T., Schwind, M., Wendt, O. & Grolik, S. (2002), Coordination of supply chains based on dispositive protocols, gdansk, poland, *in* 'Proceedings of the 10th Conference on Information Systems (ECIS)'.

Sutton, R. & Barto, A. (1998), *Reinforcement Learning: An Introduction*, MIT-Press, Cambrige, MA.

Walsh, W. E. & Wellman, M. P. (1999), Modeling supply chain formation in multiagent systems, *in* 'Agent Mediated Electronic Commerce (IJCAI Workshop), Michigan', pp. 94–101.

Walsh, W. E., Wellman, M. P., Wurman, P. R. & MacKie-Mason, J. K. (1998), Some economics of market-based distributed scheduling, *in* 'Proceedings of the Eighteenth International Conference of Distributed Computing Systems, Michigan', pp. 612–621.

Watkins, C. J. (1989), Learning from Delayed Rewards, PhD thesis, Cambridge University, Cambridge, MA.

Zhang, W. & Dietterich, T. G. (1995), A reinforcement learning approach to job-shop scheduling, *in* 'Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)', Morgan Kaufmann, Orlando, FL, pp. 1114–1120.

Zhang, W. & Dietterich, T. G. (2000), 'Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling', *Submitted to the Journal of Artificial Intelligence Research* .

Zweben, M., Daun, B. & Deal, M. (1994), Scheduling and rescheduling with iterative repair, *in* M. Zweben & M. S. Fox, eds, 'Intelligent Scheduling', Morgan Kaufmann, Orlando, FL, chapter 8, pp. 121–140.