shopST: Flexible Job-Shop Scheduling with Agent-Based Simulated Trading

Frank Yukio Nedwed[1], Ingo Zinnikus[2], Maxat Nukhayev[2], Matthias Klusch[2], and Luca Mazzola[2]

[1] Saarland University, Saarbruecken, Germany
s9frnedw@stud.uni-saarland.de
[2] German Research Center for Artificial Intelligence (DFKI), Saarbruecken, Germany
{FirstName.LastName}@dfki.de

Abstract. Paradigms in modern production are shifting and pose new demands for optimization techniques. The emergence of new, versatile, reconfigurable and networked machines enables flexible manufacturing scenarios which require, in particular, planning and scheduling methods for cyber-physical production systems to be flexible, reasonably fast, and anytime. This paper presents an approach to flexible job-shop manufacturing scheduling with agent-based simulated trading, called shopST. Aspects of real manufacturing scheduling problems form the basis for a physical decomposition of the planning system into agents. The initial schedule created by the agents in shopST through reactive negotiation is successively improved through the exchange of resource binding constraints with an additional market agent. shopST is evaluated in comparison to selected other different solution approaches to flexible job-shop scheduling.

Keywords: agents, simulated trading, flexible job-shop scheduling

1 Introduction

Modern production facilities are increasingly relying on networked machines for their benefits caused by increased flexibility and the ability for self organization. In order to further enhance economic factors, scheduling methods are needed, that take advantage of these features and can cope with the rising amount of complexity. Flexible job-shop scheduling (FJSS) is an extension of the classical job-shop scheduling problem, which is NP-hard and among the hardest combinatorial optimization problems [1]. There are several different types of solution methods available, though most of them disregard some constraints in order to simplify the problem or only regard a single cost function, e.g. makespan. The combination of several criteria or additional constraints generalizes the problem and further enhances its complexity. There is a wide range of approaches for using multi-agent systems in manufacturing in general and for job-shop scheduling in particular [20, 11, 12, 19, 17, 25, 2, 16]. In this paper, we present a novel approach, shopST, that applies agent-based distributed simulated trading [3] to

solve dynamic FJSS problems. In particular, shopST complements locally optimizing reactive agent scheduling with long-term planning via simulated trading. The results of a comparative experimental evaluation revealed that shopST is competitive in highly flexible manufacturing environments with multi-purpose machines.
The remainder of the paper is structured as follows. Section 2 shortly introduces the problem of flexible job-shop scheduling, and gives an overview of the solution and its implementation. Section 3 presents the comparative performance evaluation results, while related work is briefly discussed in Section 4. Section 5 concludes the paper with a short summary.

2 The shopST Solution for FJSS

This section introduces the problem of flexible job-shop scheduling and the first agent-based approach that makes use of simulated trading for this purpose.[3]

2.1 Flexible Job-Shop Scheduling

The problem of flexible job-shop scheduling (FJSS), in general, is to find an optimal, valid job-shop schedule S that minimizes a cost function c (e.g. makespan) for a given configuration of jobs, operations on multi-purpose machines, and is subject to certain constraints of processing. FJSS is an extension of the classical job-shop scheduling problem. Classical job-shop scheduling solutions determine a schedule for a set of jobs on a set of machines with the objective to minimize a certain criterion subject to the constraint that each job has a specified processing order through all machines, which are fixed and known in advance. A more flexible job-shop scheduling allows, for example, one operation to be performed on one machine out of a whole set of alternative machines. In the following, the type of FJSS problems our solution approach can cope with is described in more detail.
A set $J = \{j_1, \ldots, j_n\}$ of $n \in$ *jobs*, which corresponds to factory workpieces, needs to be processed with a set $M = \{m_1, \ldots, m_p\}$ of p *machines*, while every job j_i has a number of $k_i \leq p$ *operations* $O_i = \{o_1, \ldots, o_{k_i}\}$, which have to be performed in order for the job to be completed. Performing a job j_i on a machine m_j is denoted as an operation o_{ij}, which requires the exclusive, uninterrupted use of m_j over a time period p_{ij}, called *processing time*. It is assumed that the processing time can be deterministically deduced from the system in advance. A schedule S is a bijective assignment $(S(o_i) \rightarrow (m, f_i))$ of every operation o_i to a processing machine $m \in M_i^{op}$ and a *completion date* f_i, with completion dates f_{ij} for every operation and job j. The schedule is *valid*, if all time intervals, which are assigned to a machine are free of overlaps and precedences are met among the other additional constraints to the system. Each possible schedule S

[3] The source code for this project is publicly available at `https://sourceforge.net/projects/shopst/`

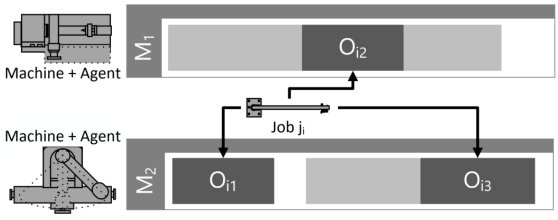


Fig. 1. Abstract example of a job-shop schedule for two machines M_1 and M_2

can be evaluated by assigning a cost c to every possible state of S via a *cost function* $c(S)$. To find an *optimal*, valid schedule then requires either to compute a valid S with minimal costs c, or to take an existing schedule and continually decrease its cost.

The following types of processing constraints are part of an extended flexible job-shop scheduling problem specification. First, any operation o_k can be performed by a number of machines M_k^{op} and it is possible that the processing time p_{ij} *varies* depending on j_i and m_j. We assume the constraint $|M_k^{op}| > 1$, which implies flexible job-shop scheduling with *multi-purpose* machines. We speak of $|M_k^{op}|$ as the factory flexibility for the remainder of this paper. Second, the schedule may also have to follow a given order of precedences for the operations to be performed. These *operation precedences* are encoded in a directed, acyclic precedence graph $G_i^{prec} = (V_i, E_i)$, where the number of vertices equals a subset of the operation set $V_i \subseteq O_i$. A directed edge $(o \rightarrow o') \in E_i$ for $o, o' \in O_i$ is part of the graph, if and only if operation o' has to be performed before operation o. In contrast to classical job-shop scheduling, the non-linear precedence constraints of the flexible version allow an arbitrary order between some processing steps of the job (e.g. drilling holes with different machines) and other precedences that are fixed (e.g. paint job only after all drilling is completed). Inflexible job-shop scheduling problems have completely linear precedence graphs. Third, possible tool changes on a multi-purpose machines may require a certain amount of time for it to prepare in between the processing of two operations. Such a *sequence-dependent setup time* s_{ikj} is the time period in which the machine cannot process any job, and which is dependent on the two operations o_{ij} and o_{kj} that shall be performed in succession. In practice, these times obey the triangle inequality $s_{ikj} + s_{kuj} \geq s_{iuj}$. Besides, jobs j_i that enter the system at a *release time* r_i cannot have their operations processed before that time, and can have a *due date* $d_i > r_i$ before which their completion is preferable, if stated in the cost function. Deadlines are mostly relevant for tardiness related cost functions like maximum lateness. We assume that already started operations o_{ij} cannot be interrupted (no preemption).

Finally, we focus on a *dynamic* version of FJSS with sequence dependent setup times and multi-purpose machines: $J\text{-}MPM|prec, r_i, d_i, sdst|c$ in the established $\alpha|\beta|\gamma$ notation for scheduling problems, whereas c denotes an arbitrary cost function [10]. The α field contains the overall class of the problem and the beta

field describes additional constraints in the setup. In particular, the sets J, M and M^{op} of FJSS problems may dynamically change during optimization of the schedule, since new jobs may enter the system, and changes to the operation sequences, machine breakdowns and other unexpected events may occur at any time. That requires the dynamic optimization to be sufficiently robust against such changes. Furthermore, the information exchange between networked machines and tools is required to be decentralized, that is, unlike most state-of-the-art solution approaches [7], we assume no global information blackboard for this purpose, as the system is decomposed by the physical constraints of the machines and not the functional ones of the algorithm.

2.2 shopST System: Overview

The proposed FJSS optimization system, shopST, consists of two phases: In the first phase, agents create a valid schedule by scheduling the resource binding constraints (operations) through standard contract net protocol based interaction in a reactive manner. In the second phase, the valid schedule is improved by the long-term schedule optimization via agent-based simulated trading. These phases are executed in succession whenever an unanticipated event disrupts the validity of the planning. The first phase creates a valid solution, which is improved in the second phase.
The use of an arbitrary short term agent-based planning system in the first phase enables local optimization of the machine schedules and a heterogeneous agent system. We used a standard contract net protocol for the local short term planning in this paper, but others can be used. The factory environment can be highly dynamic because of machine breakdowns, or other events, such that the plans have to be adapted immediately in order to commence production, which requires an anytime solution. Long-term planning with simulated trading, as first introduced by Bachem [3], is a method to find approximated solutions through several rounds of hypothetical trading between trading agents and a common market agent, followed by a consolidation round. In the following, we focus on the application of simulated trading and required modifications to fit the planning domain. An overview of the agent interaction is given in Figure 2.

Agent mapping. The trading agents are the instances in the system, which want to optimize their cost function. Thus, every machine in the system provides one trading agent. An additional non-physical market agent is existent in the network. The communication in the network is enabled via common agent technologies as described in section 2.3.
Each agent is equipped with a cost function $c(m)$, $m \in M$, that on the one hand, resembles a good evaluation of the local performance of the machine. On the other hand, the summed local costs over all agents $\sum_{m \in M} c(m)$ should be a good indicator of the overall factory performance. We experimented with different possibilities for such cost functions with varying complexity. Simple cost functions like the total operation completion date (TOpC) $c(m) = \sum f_i$ for all f_i that have a pair (m, f_i) in the schedule S or the total operation lateness (TOpL)

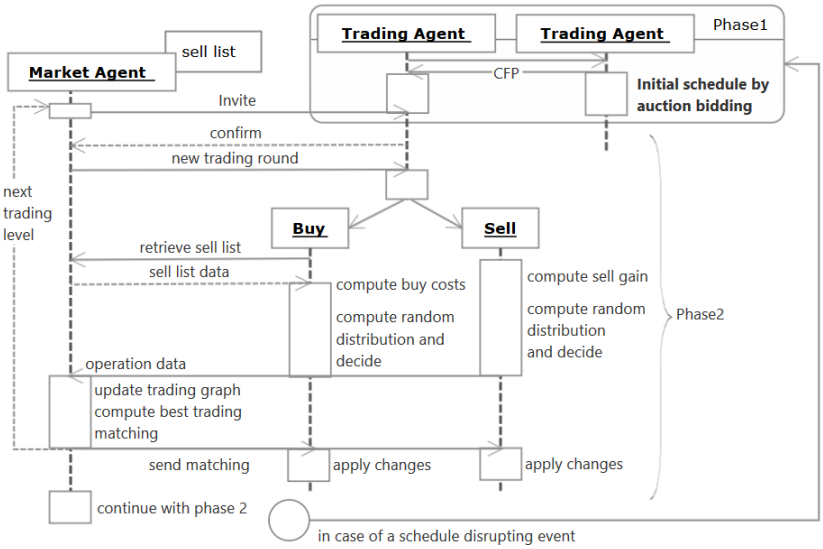


Fig. 2. Algorithm sequence and agent system structure

seem to work well and are computationally inexpensive. shopST also offers total operation tardiness (TOpT) and slack time (TOpSL) as cost functions.

Initialization. At the beginning of the simulated trading protocol, all participating agents are invited by the market agent to perform successive trading steps. In each such step, called a trading level, every agent chooses either to sell a resource binding constraint to the common market agent, or to buy such a constraint from it. Resource binding constraints of the trading machine agents in the planning domain are their processed operation plans. Whether to buy or sell is determined evenly randomized. The decision, whether a certain operation is traded or not, is not solely made in a greedy manner by local criteria, because in this case, agents would always sell the costliest operation at the moment and never buy because their resources are bound by this action. Because of this behavior, the trading is randomized and the used random distribution still depends on the anticipated buying cost or selling gain respectively, as follows. Because buying an operation from the market does almost always result in a deterioration of the local cost function, buying probabilities are derived from the difference of the cost difference the selling agent achieved, and the current buying cost. A trading agent can only buy an operation from the market if it can process this operation on the machine it is representing and successfully integrate it into its current schedule. If an operation is sold, it is deleted from the local schedule of the machine and its information is transferred to the market agent, where other agents can see and possibly buy it in upcoming trading levels. The trading agents decide which operation to trade by a random distribution depending on the impact on their local cost function. This random function is designed in a

way, that operations, that highly impact the cost function of the agent are more likely to be sold. In order to avoid stagnation, every operation has a strictly positive probability.

Trading graph. In the previous phase, it is possible for an operation to be bought by multiple agents or that sold operations are not included in the plan again. This means that the hypothetical schedule resulting after a certain amount of trading levels is not necessarily valid. In order to generate a valid schedule of lower cost, a trading graph is maintained during the execution of the trading phase. The trading graph is a bipartite graph, its vertices are represented by the single buy and sell actions. Edges link the actions belonging to the same operation and are weighted with the cost difference achieved by this trade. An exemplary trading graph is represented in Figure 3. Every node is annotated by the number of the trading level it was performed in. This results in a unique identification using trading agent and trading level, as every agent trades exactly one operation per level.

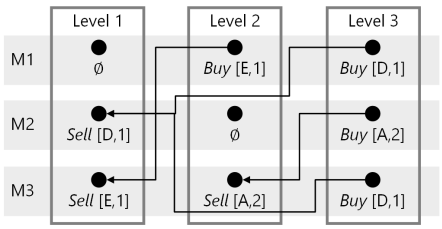


Fig. 3. Trading graph example for machine agents M1 to M3. Jobs are referred to by letters, their operations are numbered.

Trading match. In order to get to a valid schedule again, a so called trading match has to be found. This matching is a subset of the trading graph and has to satisfy the following conditions:

- A sold operation may only be bought by exactly one agent, this property is equivalent to a matching graph.
- If a vertex of round i is part of the matching, then every vertex of the same trading agent with level smaller than i has to be part of the graph, too.
- The overall weight of the graph has to be negative, which means that if the trading actions belonging to the graph are all executed, the overall cost of the system is decreasing.

In Figure 4, two trading matchings with three trading levels are displayed, which may result from the trading graph in Figure 3. **Consolidation and anytime feature.** If a trading match is found, the according trading actions are communicated to the corresponding agents. Because of the structure of the trading matching, the resulting schedule is valid and the overall costs decrease; this concludes the trading round. As each round generates another valid schedule and

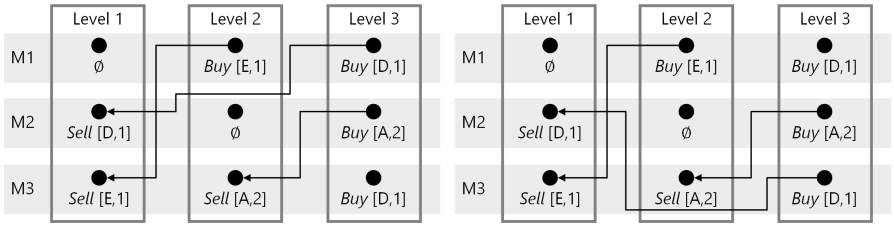


Fig. 4. Matching graphs of the trading graph in Figure 3

costs do not deteriorate after a round, the system can take new properties of the factory into account after each round. Algorithms that behave in such a way belong to the class of anytime algorithms, as they can be interrupted arbitrarily and still deliver a valid result, which is at least as good as the initial state.

Incorporated aspects of simulated annealing. A main property of simulated annealing is the acceptance of system states with higher costs [18]. In order to avoid early stagnation of the shopST algorithm, we adopted aspects of simulated annealing. Several simulated trading rounds are clustered into a super round. In a single super round, the quality of the schedule may also decrease by accepting trading matchings with positive weight up to a certain limit imposed by the temperature. The temperature decreases from round to round, similarly to the original simulated annealing meta-heuristic. In order to not affect the reactivity of the system, the sizes of these super rounds have to be adapted to the frequency of disturbances of the system. For the remainder of this paper, we will refer to the number of trading rounds in a single super round as round size.

2.3 Implementation

The shopST system has been implemented in Java. For the reactive agent planning, we used standard contract net protocol based interaction between the agents. The agent framework was built according to FIPA standards [5] and uses ACL messages to communicate between agents. For the transference of information and to keep the system generic, an ontology was used, which was specifically designed for this task. The transferred information is especially relevant for the trading agents to compute whether they can handle a workpiece operation and if they do, at which cost. The search for an optimal matching graph during the consolidation phase is computationally costly and mainly contributes to the overall runtime of the algorithm. However, its costs can be transferred into the network by the market agent and thus, make use of convenient resources as they are not bound to a specific physical instance.

3 Comparative Performance Evaluation

Experimental setting. The experimental evaluation of shopST was run on a laptop with Intel Core i5-5300U CPU@2.3 GHz processor. In order to test

shopST performances, we pragmatically determine some optimal values for its main parameters first. The main metrics used for the comparison are the quality of the produced solutions and the effects the factory flexibility has on solutions. As the criterion of solution quality, the total length of the computed schedule, i.e. the makespan, has been taken. Regarding the testing of flexibility, we adopted its definition from [14], i.e. the average number of machines that can execute a given operation, and the best makespan reached as a measure for different settings of this parameter. The solution quality of shopST was compared with that of the most recent and successful FJSSP solving algorithms: HTSA [21], Zhang GA [33], AIA [4], X2010 [31], MOGA [28], P-DABC [23], X2009 [30], MOPSO [13], and HSFLA [22]. Whenever available, experimental results from the original papers were reused. The Zhang GA had to be re-implemented due to the unavailability of the original code, in order to run it on the same infrastructure and to support more detailed comparisons with shopST. Every run was repeated five times on the same instance, in order to obtain meaningful and comparable results. For the makespan analysis, the best result was adopted, in accordance with the general approach reported by the compared algorithms; for flexibility analysis, the results were averaged to overcome the non-deterministic nature of the shopST algorithm.
Three popular collections of problem instances have been used for testing, namely:

1. Kacem [15]: 4 problems with total flexibility and different number of operations per job; every operation can be processed on any one of the machines. The number of machines ranges from 5 to 10, number of operations from 12 to 56.
2. Brandimarte [6]: 10 problems, which were randomly generated using a uniform distribution between two given units. The number of jobs ranges from 10 to 20, number of machines from 4 to 15, number of operations per job from 3 to 103 and number of machines per operation from 2 to 6.
3. Hurink et al.[14]: 129 test problems divided by flexibility levels into sdata, edata, rdata and vdata subsets. The number of jobs ranges from 6 to 30 and the number of machines ranges from 5 to 15.

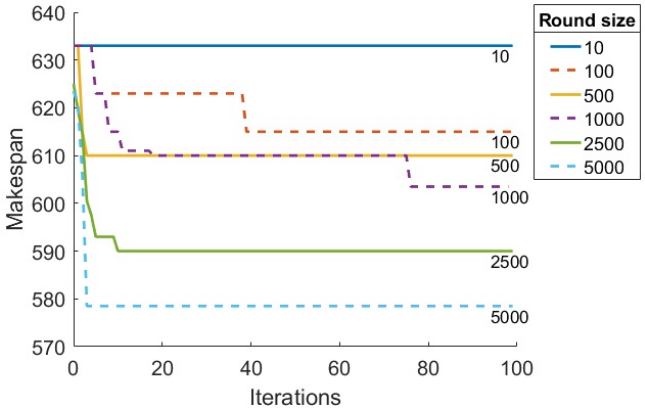
Based on the number of operations, machines and flexibility, the problem instances from Brandimarte [6] and Hurink [14] datasets were grouped as specified in Table 1. The problems were arranged by the number of machines and operations into three groups (small, average, and large). Each group was split further by its flexibility level into two subgroups with low and high flexibility. The *small* size group, *high flexibility* sub-group presents only a single instance, as only one small highly flexible problem is included in the aforementioned datasets. The test problems were chosen to cover to a certain extent the full data space for every defined subgroup. It is worth to be noted that based on the very limited extension of the Kacem dataset (4 problems) and its full flexibility, the set contains only outliers with respect to the classification dimension and consequently no representative of it was selected, as for Table 1.
The solving of each problem listed in Table 1 has been tested with different values of round size, ranging from 1 to 5000. Based on the experimental results

Size	Flexibility level	Problem	Operations	Machines	Flexibility
small	low	Mk01	55	6	3
		la05(vdata)	50	5	2.5
		la06(rdata)	75	5	2
	high	Mk02	58	6	6
average	low	Mk04	90	8	3
		la11(rdata)	100	5	2
		la25(rdata)	150	10	2
	high	Mk07	100	5	5
		la18(vdata)	100	10	5
		Mk03	150	8	5
large	low	la26(rdata)	200	10	2
		la36(rdata)	225	15	2
		la31(rdata)	300	10	2
	high	la36(vdata)	225	15	7.5
		la26(vdata)	200	10	5
		Mk09	240	10	5

Table 1. Selected problems grouped by size and flexibility from [14] and [6]

shown in Table 2, one can see that there is a direct correlation between the problem size and the round size: larger problems require larger round sizes. A comparison of flexibility levels shows that more flexible problems require more rounds to converge, which can be explained by the higher number of trade options for the agents. The result of an example run of such complex, highly flexible problems from la26(vdata) is shown in Figure 5 in which, for readability reasons, the range of round size is divided into representative discrete values (10/100/500/1000/2500/5000).

Fig. 5. Makespan convergence of la26(vdata) problem with different round sizes



Size	Flexibility level	Optimal round size
small	low	100
	high	100-500
average	low	1000-2500
	high	2000-5000
large	low	2500-5000
	high	5000

Table 2. Optimal round size by size and flexibility

Based on experiments, the other parameters of shopST have been chosen as five trading levels, 100 super rounds and TOpC as cost function. The initial solution for shopST has been generated by randomly assigning operations to suitable machine agents.

Solution quality. The solution qualities produced by all tested algorithms including shopST for different datasets are shown in tables 3 and 4. As a result, shopST produces a solution quality that is comparable to that of the selected representative state-of-art solution algorithms although it has not found the best solutions most of the time. The last rows of these tables show the relative deviation with respect to shopST. The relative deviation for each problem instance is defined as

$$dev = [(MK_{comp} - MK_{shopST})/MK_{shopST}] * 100\%$$

where $\mathrm{MK_{shopST}}$ is the makespan obtained by shopST and $\mathrm{MK_{comp}}$ is the average makespan of all the other algorithms shopST is compared to. As a result, shopST underperformed by an average of 13.5% (ranging from 0% to about 25%). Zhang GA[33] found 8 out of 10 best solutions and for this reason was chosen for a more detailed comparison with shopST. Please keep in mind that the notion of iteration differs for shopST and Zhang GA[33]: While in Zhang GA one iteration is one evolution of the population and takes around 60 msec, in shopST one iteration corresponds to one super round of simulated trading, which can run from several seconds to several minutes depending on the round size.

Execution time. In order to compare the runtimes of shopST and Zhang GA, both algorithms have been executed with optimal parameters on the la40(vdata) problem instance. The experiment was run 10 times for each algorithm, and the best results were selected. The results, as shown in Figure 6 for shopST and in Figure 7 for Zhang GA, reveal that the execution time of shopST is three orders of magnitude larger than that of Zhang GA and appears to be connected with the high round size requirement for reaching an optimal solution by shopST.

Flexibility. For the second evaluation metric, the effects of problem flexibility on solution quality were addressed in the following experiment: shopST and Zhang GA were executed on problems with different degrees of flexibility. In order to simulate different levels of flexibility, an original non-flexible problem la40(sdata) from Hurink dataset was modified by the application of a new parameter P, representing the probability that a particular machine can execute

Fig. 6. Execution time of shopST

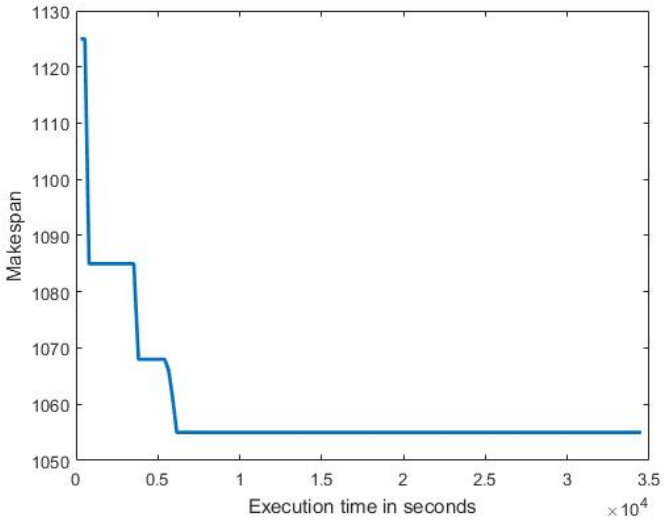
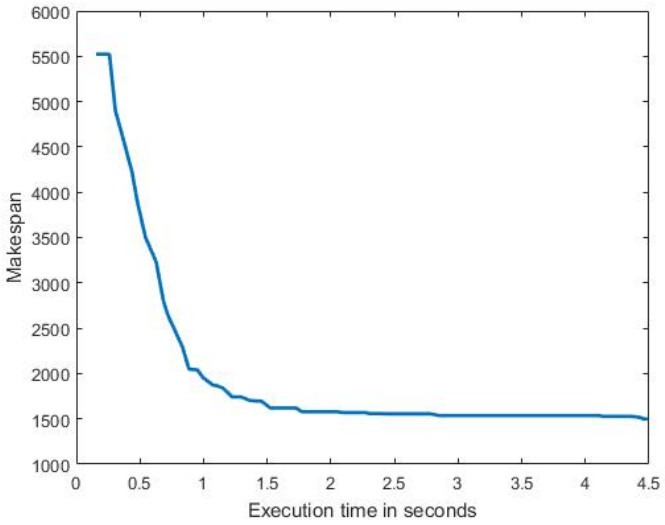


Fig. 7. Execution time of Zhang GA [33]



Algorithms	Instance 1	Instance 2	Instance 3
shopST	12	**7**	13
Zhang [33]	**11**	**7**	**11**
HTSA [21]	**11**	**7**	**11**
AIA [4]	-	**7**	**11**
Xing [31]	12	**7**	**11**
MOGA [28]	**11**	**7**	**11**
P-DABC [23]	**11**	**7**	**11**
MOPSO [13]	**11**	**7**	**11**
dev(%)	-6.9	0.0	-15.4

Table 3. Makespan results for Kacem[15] data, best solutions in bold

Algorithms	MK01	MK02	MK03	MK04	MK05	MK06	MK07	MK08	MK09	MK10
shopST	47	34	229	84	196	80	164	558	342	267
Zhang [33]	**40**	**26**	**204**	**60**	173	**58**	144	**523**	**307**	**198**
Xing[30]	42	28	**204**	68	177	75	150	**523**	311	227
MOGA [28]	**40**	**26**	**204**	66	173	62	**139**	**523**	311	214
HTSA [21]	**40**	**26**	**204**	61	**172**	65	140	**523**	310	214
HSFLA [22]	**40**	**26**	**204**	62	173	64	141	**523**	311	215
AIA [4]	**40**	**26**	**204**	**60**	173	63	140	**523**	312	214
MOPSO [13]	**40**	**26**	**204**	61	173	62	**139**	**523**	310	214
dev(%)	-14.3	-22.7	-10.9	-25.5	-11.5	-19.8	-13.5	-6.3	-9.3	-20.0

Table 4. Makespan results for Brandimarte[6] data, best solutions in bold

a particular operation. This is to simulate flexible manufacturing environments with multi-purpose machines. The results shown in Figure 8 reveal that shopST significantly improves its solution quality for more flexible problems, and outperforms Zhang GA in this regard. In particular, Zhang GA shows a decrease in its performance with increasingly flexible problems, as depicted in Figure 9. Allowing more machines to execute particular operations results in an increase in the problem search space, that increases the probability for Zhang GA to be stuck in a local minimum, hindering its capability of converging to a globally optimal value. ShopST, on the other hand, works solely on flexible problems, because exchanges between agents are only enabled if multiple machines can exchange operations. As a consequence, a more flexible problem enables a larger number of exchange points and therefore the performance of shopST greatly improves with a greater flexibility of the multi-purpose machines.

One main strength of shopST is that it excels in solving highly flexible JJS problems with agent-based simulated trading. Besides, it natively adapts online to dynamic events that affect the problem that is currently being solved without the need of a full restart. These advantages, however, come at the cost of a comparatively higher execution time. Overall, shopST can be considered as a valuable solution for job-shop scheduling in highly flexible and dynamic cyber-physical production systems and environments, if there are no hard time constraints for solution availability.

Fig. 8. Results of flexibility test on shopST

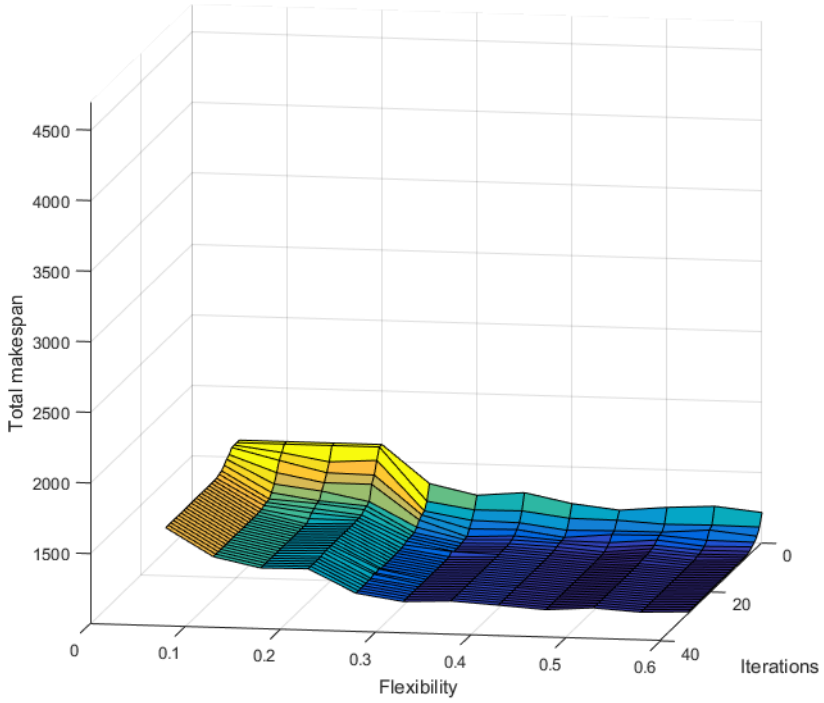
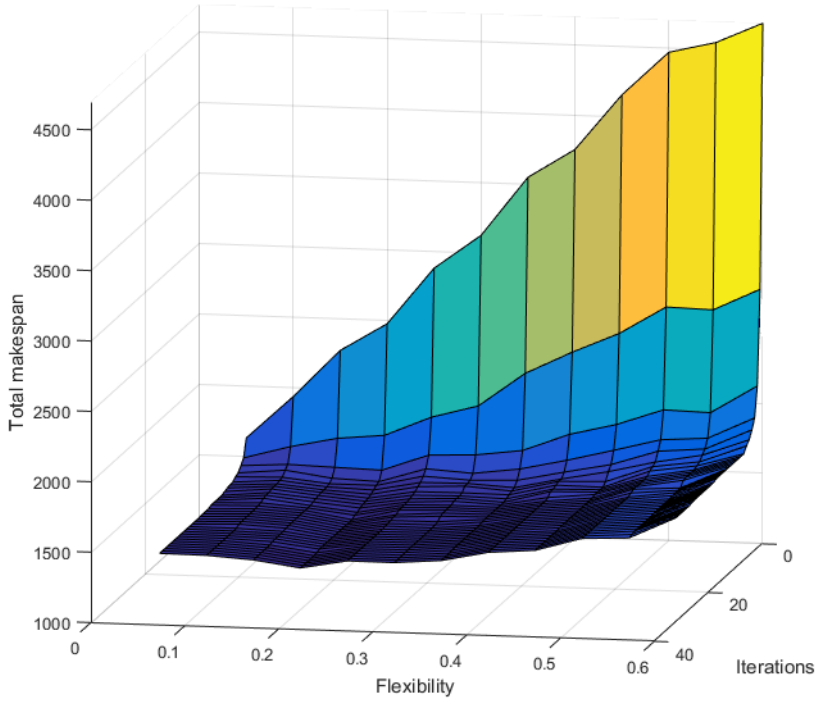


Fig. 9. Results of flexibility test on Zhang GA [33]



4 Related Work

For the comparative performance evaluation of shopST, we selected different types of state-of-the-art FJSS problem solving approaches including multi-agent system based ones. The solutions qualities of shopST are close to those of these approaches, which utilize genetic algorithm, artificial immune, knowledge-based ant colony optimization, Pareto-based discrete artificial bee colony, modified discrete particle swarm optimization, shuffled frog-leaping, hybrid tabu search. Of course, there are many other agent-based approaches for dynamic and distributed job-shop scheduling in manufacturing [20, 11, 29, 12, 19, 32, 17, 2]. For example, [29] presents an actor-based approach to job-shop scheduling using Lagrangian relaxation which may adapt its schedule after dynamic events quickly but no values are given for comparison. BnB-ADOPT [32] is a memory-bounded asynchronous distributed constraint optimization problem solver that uses the agent framework of ADOPT. It performs exceptionally well in regard to runtime and solution quality but, in contrast to shopST, the dynamic constrained optimization problem description has to be explicitly encoded for every agent in prior. However, to the best of our knowledge, shopST is the first agent-based approach with simulated trading used to solve the class of FJSS problems defined above. From the results of the comparative experimental testing of flexibility it became evident that shopST has its general strength in highly flexible manufacturing environments with multi-purpose machines.
Scheduling approaches can be characterized as constructing a schedule vs. optimizing a given schedule. In the first case an (ideally) exact solution for a given problem is generated (cf. [24] for a thorough overview of classical approaches). Optimization approaches improve an already existing schedule with respect to a cost function and are in general based on heuristics or meta-heuristic procedures and generate solutions iteratively, at the expense of non-optimal schedules [27]. A related field is online scheduling [26], where information about the problem domain is restricted (e.g. incoming jobs are only known when they arrive and processing times only after a job is completed). shopST addresses the problem of the optimization and repair of schedules in flexible and dynamic manufacturing environments with multi-purpose machines. Closely related approaches are based on (meta-)heuristics, since standard algorithms assume complete knowledge about the problem domain which usually implies a restart of the algorithm after a change of the problem domain. In recent years, a number of job-shop scheduling approaches based on meta-heuristics have been proposed for this purpose. [6] and [14] used tabu search for solving the FJSS problem, while [8] combine approaches using tabu search with simulated annealing. Several approaches for the JSS and FJSS based on evolutionary algorithms have also been developed (for a survey see e.g. [9]). Genetic algorithms such as those developed by Zhang [33] or Xing [30], for example, are an efficient way for schedule optimization. However, they have two major drawbacks: The first is that their information structure is functional and does not take advantage of an underlying agent encapsulation. The other is that they lose performance and solution quality in more flexible factory layouts, a use case which gets more and more common.

5 Conclusions

We presented a novel approach, shopST, that applies agent-based simulated trading to solve dynamic FJSS problems. shopST complements locally optimizing reactive agent scheduling with long-term optimization of the valid schedule via simulated trading. The results of a comparative experimental evaluation revealed that shopST is particularly competitive in highly flexible manufacturing environments with multi-purpose machines. Future work includes further investigation of robustness against disruptive events and performance trade-offs compared to other negotiation-based approaches when applicable to the same problem.

Acknowledgements. The work described in this paper was partially funded by the German Federal Ministry of Education and Research (BMBF) in the project INVERSIV and the European Commission in the project CREMA.

References

1. J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3), 1988.
2. M. E. Aydin and E. Oeztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2):169–178, 2000.
3. A. Bachem, W. Hochstättler, and M. Malich. The simulated trading heuristic for solving vehicle routing problems. *Discrete Applied Mathematics*, 65(1-3), 1996.
4. A. Bagheri et al. An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems*, 26(4), 2010.
5. F. Bellifemine, A. Poggi, and G. Rimassa. Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99. London, 1999.
6. P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3), 1993.
7. P.R. Cohen, A. Cheyer, M. Wang, and S.C. Baeg. An open agent architecture. In *AAAI Spring Symposium*, volume 1, 1994.
8. P. Fattahi, M.S. Mehrabad, and F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Intelligent manufacturing*, 18(3), 2007.
9. M. Gen and L. Lin. Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. *Intelligent Manufacturing*, 25(5), 2014.
10. R.L. Graham et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5, 1979.
11. Nouri H.E., Driss O.B., and Ghedira K. A classification schema for the job shop scheduling problem with transportation resources: State-of-the-art review. In *Proceedings of Artificial Intelligence Perspectives in Intelligent Systems*.
12. C.Y. Hsu, B.R. Kao, and K.R. Lai. Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling. *Engineering Applications of Artificial Intelligence*, 53:140–154, 2016.
13. S. Huang et al. Multi-objective flexible job-shop scheduling problem using modified discrete particle swarm optimization. *SpringerPlus*, 5(1), 2016.

14. J. Hurink, B. Jurisch, and M. Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4), 1994.
15. I. Kacem, S. Hammadi, and P. Borne. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and computers in simulation*, 60(3), 2002.
16. P. Kapahnke, P. Liedtke, S. Nesbigall, S. Warwas, and M. Klusch. Isreal: an open platform for semantic-based 3d simulations in the 3d internet. In *Proc. International Semantic Web Conference, LNCS 6414*. Springer, 2010.
17. A. Karageorgos, N. Mehandjiev, G. Weichhart, and A. Hämmerle. Agent-based optimisation of logistics and production planning. *Engineering Applications of Artificial Intelligence*, 16(4), 2003.
18. S. Kirkpatrick et al. Optimization by simulated annealing. *Science*, 220(4598), 1983.
19. P. Leitao. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22:979–991, 2009.
20. P. Leitao et al. Smart agents in industrial cyber-physical systems. *Proceedings of the IEEE*, 104 (5):1086–1101, 2016.
21. J. Li, Q. Pan, and Y.C. Liang. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 59(4), 2010.
22. J. Li, Q. Pan, and S. Xie. An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems. *Applied Mathematics and Computation*, 218(18), 2012.
23. J.Q. Li, Q. Pan, and K.Z. Gao. Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. *Advanced Manufacturing Technology*, 55(9), 2011.
24. M. Pinedo. *Scheduling. Theory, Algorithms, and Systems*. Springer, 2016.
25. D. Pooja and S. Joshi. Auction-based distributed scheduling in a dynamic job shop environment. *Production Research*, 40(5):1173–1191, 2002.
26. K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.
27. A. Rossi and G. Dini. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23(5):503–516, 2007.
28. X. Wang et al. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *Advanced Manufacturing Technology*, 51(5), 2010.
29. G. Weichhart and A. Hämmerle. Multi-actor architecture for schedule optimisation based on lagrangian relaxation. In *German Conference on Multiagent System Technologies*. Springer, 2016.
30. L.N. Xing, Y.W. Chen, and K.W. Yang. An efficient search method for multi-objective flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 20(3), 2009.
31. L.N. Xing et al. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10(3), 2010.
32. W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. IFMAS, 2008.
33. G. Zhang, L. Gao, and Y. Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4), 2011.