Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

A SIMULATION-BASED OPTIMIZATION HEURISTIC USING SELF-ORGANIZATION FOR COMPLEX ASSEMBLY LINES

Evangelos Angelidis
Daniel Bohn
Oliver Rose

Department of Computer Science
University of the Federal Armed Forces Munich
85577 Neubiberg, GERMANY

ABSTRACT

Our paper deals with the scheduling of complex assembly lines with a focus on Job Shop Scheduling Problems that exhibit several assembly specific characteristics: many isolated project networks with precedence constraints and thousands of jobs, time bound requirements for jobs and projects, limited resources with individual scheduling and resource lock rules. Formally it is defined as a Multi-Mode Resource-constrained Multi-Project Scheduling Problem with splitting activities. Problems that display these characteristics are often difficult to solve with classical scheduling approaches within acceptable runtime. Simulation-Based Optimization offers an auspicious manner of dealing with those domain specific problems. Using this approach we present a decentralized heuristic evident in self-organization in nature. Typical algorithms attempt to solve the above problems globally. In our solution, the jobs of the network take over the active role. They communicate with their neighbors and the allocated resources, each having the local goal to optimize their own situation.

1 INTRODUCTION

This paper explores solution approaches for scheduling problems in complex assembly lines in industrial environments. Presently the global market demands reduced production costs and on time delivery, especially in assembly lines with workforce planning. Our research focuses on small series or even unique items such as turbines, planes or industrial machines. A scenario contains various products with different production plans (also referred to some authors as projects (Pinedo 2007; T'kindt 2006) or networks (Brucker and Knust 2006; Pinedo 2008)), precedence constraints and thousands of activities. Many elements, such as activities and products, have many time bound requirements. The activities do not have a fixed processing time but many possible modes, in which their time is closely linked to the resources assigned to it. In addition, the production has limited resources with individual scheduling and finite buffers. The scheduling of this type of assembly production lines is a very complex task even for experienced foremen.

The scheduling of such complex assembly lines is a large combinatorial optimization problem and is often mentioned in the literature as a Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MMRCMPSP) with activity splitting (Angelidis, Naumann, and Rose 2012). Unfortunately this is NP-hard and real-life problems of this size cannot be solved with classic approaches in short runtimes. The research in this area is mainly based on small or medium sized problems, which are often solved by exact approaches or genetic algorithms. These strategies are not suitable for our large real-life problems, especially with respect to short runtimes. Pappert, Angelidis, and Rose (2010) give a detailed explanation of this scheduling domain and its plausible solution strategies. They also present a promising alternative that produces efficient solutions for real-life problems in short runtime (Simulation-Based Optimization approach). The basic idea is to simulate a scenario, analyze it and create a new one according to all previously simulated scenarios. The aim of these iterations is to generate a feasible solution with a minimum number of steps.

In this paper we focus on solutions for such problems, working on the Simulation-Based Optimization approach without using a central control or a global optimization function; instead we developed a heuristic decentralized algorithm based on self-organization. Active subunits of the problem description have been defined, which try to evaluate their own state and goal function during their interaction with each other. This is the main principle of self-organized systems, in which the active members either act together or against each other changing the system dynamically and attempting to balance towards an optimized scenario.

In Section 2 we will discuss Simulation-Based Optimization in more detail. In Section 3 we present the modeling approach and the optimization heuristic based on self-organization. Section 4 provides a short overview of the experiments and the results. In Section 5 we draw certain conclusions and discuss future research.

2 SIMULATION-BASED OPTIMIZATION APPROACH

Almost 16 years ago, Shapiro (1996) presented a paper, which was using Simulation-Based Optimization (SBO) for a G1/G/1 problem specification. In the years that followed with the help of faster PCs and improved heuristic optimization search it became possible for simulation tools to integrate optimization packages for bigger problems (Law and McComas 2002). April et al. (2003) presented a practical introduction about SBO and discussed the metaheuristic approach to it. From that point onwards that approach is more often used in commercial tools and not just solely for problems of academic research. The optimization of our domain of interest needs solution strategies, which can handle huge search spaces and find suitable scenarios in short runtimes. An interesting concept for Simulation-Based Optimization is illustrated in Figure 1.

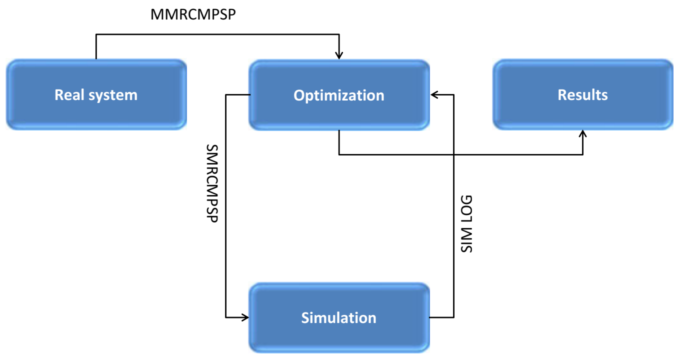

Figure 1: Concept of the Simulation-Based Optimization.

At the beginning the real system is transformed to a MMRCMPSP model. This model describes the manufacturing facilities and contains all the required information to create all future possible scenarios of this assembly production line. The optimizer, according to the solution strategy and the results of all already simulated scenarios (SIMLOGs), creates new scenarios trying to find a feasible solution. The scenarios are classified as a Single-Mode Resource-Constrained Multi-Project Scheduling Problem (SMRCMPSP) with activity splitting.

2.1 Simulation-Based Optimization Platform

Pappert, Angelidis, and Rose (2010) presented a framework for simulation-based scheduling of assembly lines. We improved this concept, made broader applications and created a new tool called the Simulation-Based Optimization Platform with the following components.

2.1.1 Runtime Environment, Communication Between Components

The new platform is based on OSGi (OSGi Alliance 2012) as a runtime environment. We chose OSGi because it offers many specifications, associated implementations and it is state of the art in software development. So it is possible that the platform components can communicate per event (sync or out of sync) or via direct service call between each other. The existing remote interface specification is also crucial to our research due to the fact that it allows the extension of our platform with remote interface in short time and with minimal effort.

2.1.2 Metamodel Generation and Interface Extensions

In addition to the framework, the platform should also address different application areas. With respect to this constraint we decided to provide a general interface to this platform based on the Eclipse Modeling Framework (Steinberg et al. 2008). The application of EMF has also the added benefit, which allows us to flexibly modify our current meta data model while working with several companies of different size and assembly approaches. With the modeled interface it is possible to deploy the generated code directly in our platform. Additionally OSGi allows multiple data models in one runtime environment. With the use of this interface, the user is able to easily implement all platform components.

2.1.3 Model and Result Administration

The framework manages the model and the results by storing them as files. The new platform uses for persistence a scalable, high-performance database, the mongoDB (10gen 2012). It is a new database technology, which is not based on relational database design. The advantages of this approach are optimized persistence, shorter access times and more flexible access management.

2.1.4 Visualization

Web technology and cloud services are trend-setting these days. Thus, we decided to provide an intuitive web front-end as it is more user-friendly to upload factory plans, trigger optimization runs with different parameters and visualize the optimization results.

2.2 Optimization of Complex Assembly Lines

Further goals for our optimization are to find models in short runtime which have production plans with minimized slack, balanced workforce with minimal amount of elements and concurrently highest resource utilization. All these goals interact with each other thus making the search for an optimal solution more challenging. Noack and Rose (2008) tackle similar problems also by the application of Simulation-Based Optimization.

A detailed description of our manufacture and simulation model can be found in Pappert, Angelidis, and Rose (2010), Angelidis, Pappert, and Rose (2011). In the following section a summary of a simulation model is presented. The main objects are orders that contain production plans which contain a network of activities. Orders, production plans and activities have adjusted earliest possible start and latest possible end dates in order to be prioritized for better results. Production plans are defined as activity-on-arc (AOA) networks with alternative production paths. Activities have a selected mode with a current number of assigned resources and a processing time. Resources in the simulation model have schedules chosen from the defined schedule set in the manufacturing model. The model has a list of dispatching rules and other settings to run the model according to the requirements of every optimization method.

The SIMLOG contains all the required information to describe a simulated scenario, for instance the states of the resources and activities, with the goal to evaluate it. For example in Figure 2 we show a time-line of an executed activity. The activity time-line provides the opportunity to carry out a detailed study of the activity's behavior. Scenarios that have activities with violated due dates are not solution candidates. For instance an activity cannot start before its earliest start date. This date is unchangeable, but the optimizer can set an adjusted earliest start date to affect the production flow. When this date is reached, the activity can only start if all required predecessors are finished. The time slice with the state waiting for constraints is an indicator for an unnecessary delayed activity, which can impact the ability to meet its due date and that of others. For the same reason the time slice with the state waiting for resources is an indicator of that although the activity has no constraints any more. A possible low priority of the activity or a high utilization of the required resource affects the start. When the activity has finished before the latest end date has been adjusted, is an indicator that the due date has not been violated; it also suggests an increase of the processing time by choosing a mode with less resources with an aim to lower the slack and cost of the activity. A more detailed explanation of the simulation model and a way to simulate it, can be found in Angelidis, Pappert, and Rose (2011).

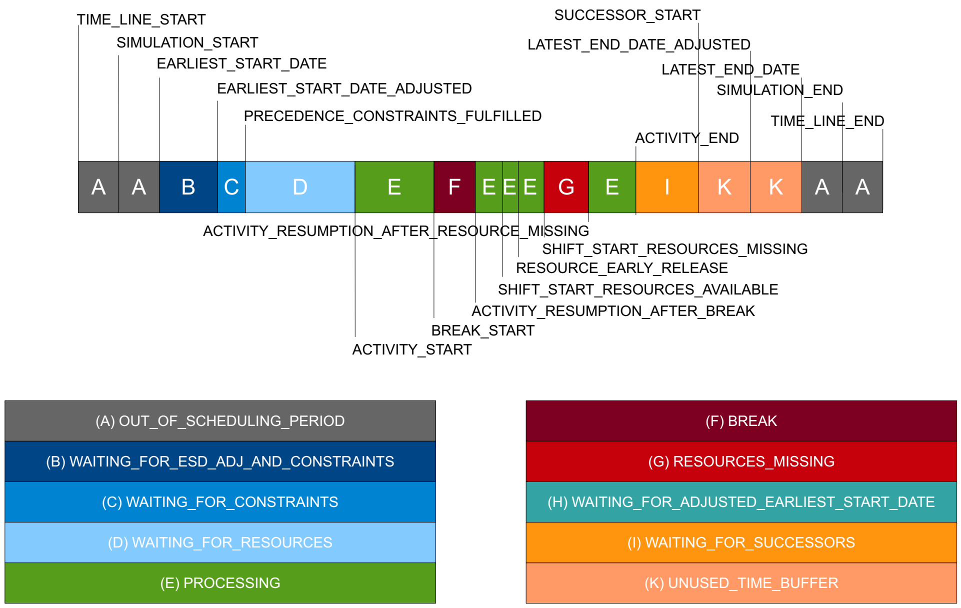
TIME_LINE_START
SIMULATION_START
EARLIEST_START_DATE
EARLIEST_START_DATE_ADJUSTED
PRECEDENCE_CONSTRAINTS_FULFILLED
SUCCESSOR_START
LATEST_END_DATE_ADJUSTED
LATEST_END_DATE
SIMULATION_END
TIME_LINE_END
ACTIVITY_END
A A B C D E F E E E G E I K K A A
ACTIVITY_RESUMPTION_AFTER_RESOURCE_MISSING
SHIFT_START_RESOURCES_MISSING
RESOURCE_EARLY_RELEASE
SHIFT_START_RESOURCES_AVAILABLE
ACTIVITY_RESUMPTION_AFTER_BREAK
BREAK_START
ACTIVITY_START
(A) OUT_OF_SCHEDULING_PERIOD
(B) WAITING_FOR_ESD_ADJ_AND_CONSTRAINTS
(C) WAITING_FOR_CONSTRAINTS
(D) WAITING_FOR_RESOURCES
(E) PROCESSING
(F) BREAK
(G) RESOURCES_MISSING
(H) WAITING_FOR_ADJUSTED_EARLIEST_START_DATE
(I) WAITING_FOR_SUCCESSORS
(K) UNUSED_TIME_BUFFER

Figure 2: Activity time-line in a log of simulated scenario (SIMLOG).

3 AN OPTIMIZATION HEURISTIC BASED ON SELF-ORGANIZATION

Self-organization is a frequently observed phenomenon in biological systems. A decentralized control is often the only reason for survival, especially for systems with a large number of elements and a complex communication process with each other. Every subunit tries to optimize its own function without any influence from a central unit (Seeley 2002), with access only to local information and with sole communication with its neighbors in an attempt to increase its goal function. Camazine et al. (2003) delves into the coordinated schooling of fish. It is described as a system where thousands of fish are moving together as a group without a leader. The fish have no access to global information and try to orientate themselves or find food more easily according to their neighbors' information. Based on the theory of self-organization and research on object-oriented models of fish school behavior, such as Reuter and Breckling (1994), we have devised an approach which will enable us to solve problems in our domain aforementioned in the previous chapter.

3.1 Modeling Approach

The main issue is to transform the behavior of the fish school to a model that describes production plans. An activity of a production plan can represent a fish, the active unit of the group. The production plan can represent a fish school. In our model there are more than one production plans (group of fish) and they can communicate with each other. In real systems as well as in our model, a fish has a changing neighborhood. In our model we have split the dynamic neighborhood of an activity into the following main sets:

- a dynamic set of executed immediate predecessors and successors,
- a dynamic set of all activities which have competed with each other for at least one resource (these activities can also belong to other production plans),
- the allocated resources that the activity requires in order to start processing.

In the first set we observe only the executed successors and predecessors activities, regarding to the selected alternative paths of the production. In the second set there are activities they can also belong to other production plans (although the plans are independent graphs). Communication is allowed according to the common required resources and helps to share requests, specify priorities or synchronize them. The third set contains all the resources that an activity needs and which have useful information like schedule plans and utilization.

In nature self-organized systems are dynamic systems, in which the subunits continually try to reach a maximum amount of their local goal function, while concurrently and in an indirect manner the system attempts to maintain balance. Unfortunately in our modeling approach it is not possible to model continuous change as it manifests in nature. The activities have to make decisions as a group. We have chosen a modeling approach based on the following principles depicted in (Figure 3). When an activity (fish as a part of the school) has to make a decision, it needs to communicate initially with the neighborhood (sending/getting requests). Afterwards it analyzes its own situation according to the simulation results of its last decisions, it reacts based on its optimization function and in the end it communicates again with the neighborhood (sending decisions). The communication flow follows strict rules. It starts from all the executed end activities towards the initial activities and then back to the end activities again. In other words all these synchronized actions of the activities describe a classic step of the simulation-based optimization, in which the optimization part creates a new scenario according to the already simulated scenarios to find a more suitable solution.

The activity has access to the following local information:

- the existence of an activity that exceeds its due date,
- the occurrence of a change from a mode to another or from the earliest start date in the last scenario,
- the activity time-line for the last simulated scenario (Figure 2),
- the state history of the resources that the activity needs to start (state analysis and communication with the competing activities).

3.2 Optimization Approach

Based on this general approach we can create many different goal-driven functions for the activities in order to observe the behavior of the system. We present an algorithm with the following internal goals: Firstly, every activity tries to finish before its due date. If this goal is achieved for every activity, the process tries to use as few resources as possible (longer processing time) without violating its due date (slack minimization).

At the initial step the method requires an already existing scenario for comparison reasons. Many different scenarios with different characteristics can be created as an initial starting point. We have chosen a simulation model with the following characteristics: The activities have randomly chosen modes and the minimum possible earliest start dates and maximum latest end dates. A detailed description of the manufacture

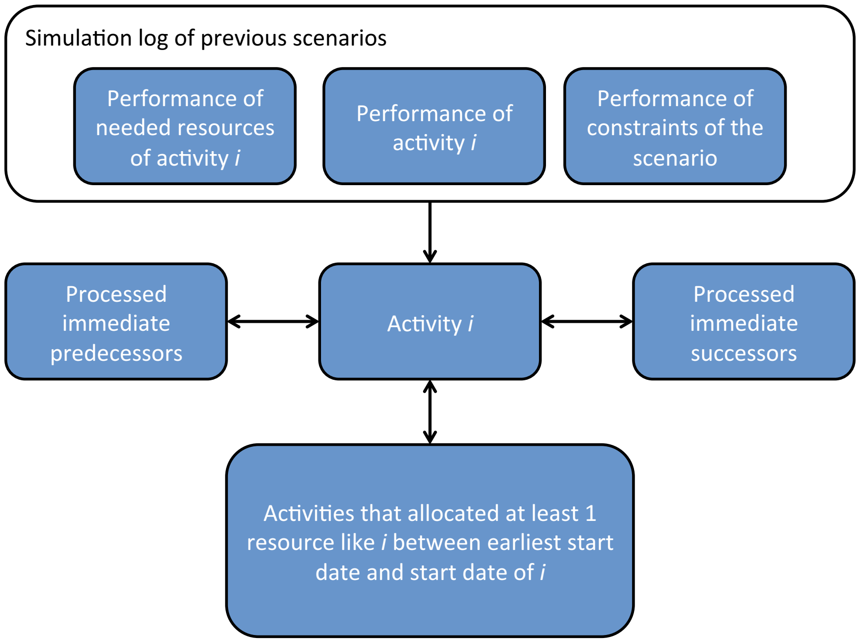
Simulation log of previous scenarios
Performance of needed resources of activity *i*
Performance of activity *i*
Performance of constraints of the scenario
Processed immediate predecessors
Activity *i*
Processed immediate successors
Activities that allocated at least 1 resource like *i* between earliest start date and start date of *i*

Figure 3: Neighborhood of the activity.

mode can be found in Pappert, Angelidis, and Rose (2010), Angelidis, Pappert, and Rose (2011). The initial activities of the graph have as earliest start date (ESD) the $\max(ESD_{act}, ESD_{myProduct}, ESD_{myOrder})$. The end activities have as latest end date (LED) the $\min(LED_{act}, LED_{myProduct}, LED_{myOrder})$. The model has a list of dispatching rules, which is chosen by the user. The creation of this list is a crucial point and the user should analyze the model well in advance, mainly because an unsuitably chosen list would have a huge negative influence on the results. More details on our self-organization method can be found in the next paragraphs.

3.2.1 Primary Goal Function - No Activity Violates Its Latest End Date

The primary goal of the algorithm is to create a scenario where no activity exceeds its due date. If an activity exists with due date violation, the algorithm tries to solve it, otherwise the secondary goal function is chosen. The executed end activities start to communicate with their neighbors backwards through all the production plans, to share requests and necessary information.

Through this backward communication, the algorithm searches for activities with due date violation. The backward communication is illustrated in Figure 4. Activities that finish after their due date have to interact with their neighbors (sending requests) to improve their situation. If their time-line does not have a negligible *waiting for constraints* time slice, this means that an earlier finish of the predecessors allows the activity to try and start earlier. The activity can avoid a possible due date violation. For that reason it sends a request with the required negative value $Req = (DueDate_{act} - FinishedTime_{act})$ to all predecessors for earlier finishing. If an activity does not exceed its due date and has not received any requests for earlier finishing, it will send to all predecessors a request $Req = 0$.

On the other hand, if it obtains any earlier finishing requests from successors $i, ..., k$ and at the same time the activity has a *waiting for constraints* time slice, then a new request will be forwarded to all the predecessors, with the negative value $min(Req_i, ..., Req_k)$. If the activity exceeds its due date then this negative value is $min(Req, Req_i, ..., Req_k)$. The idea is to forward the minimum required value with the request until a point is found to start optimizing. Through this principle, the requests reach the delay creation points.

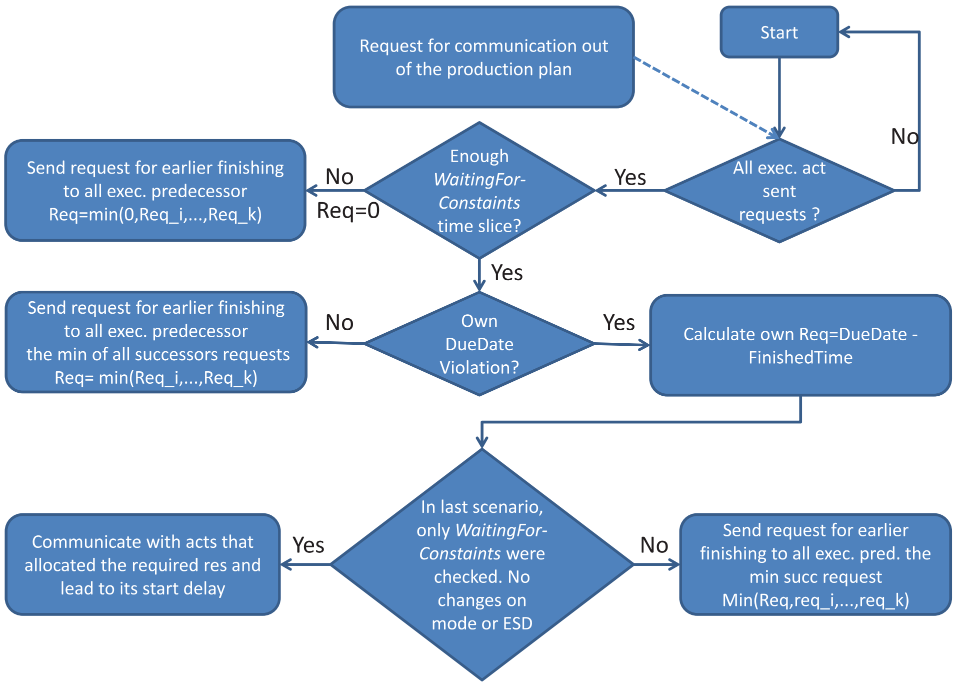
Start
Request for communication out of the production plan
All exec. act sent requests ?
No
Yes
Enough *WaitingFor-Constaints* time slice?
No
Req=0
Send request for earlier finishing to all exec. predecessor Req=min(0,Req_i,...,Req_k)
Yes
Own DueDate Violation?
No
Send request for earlier finishing to all exec. predecessor the min of all successors requests Req= min(Req_i,...,Req_k)
Yes
Calculate own Req=DueDate - FinishedTime
In last scenario, only *WaitingFor-Constaints* were checked. No changes on mode or ESD
Yes
Communicate with acts that allocated the required res and lead to its start delay
No
Send request for earlier finishing to all exec. pred. the min succ request Min(Req,req_i,...,req_k)

Figure 4: Backwards communication of primary goal function.

When the backwards communication reaches all initial activities, the communication changes direction. At this point every activity has sent its required requests. Every activity tries now to improve its situation and behave accordingly to the requests. The first activities, that were requested to finish earlier, try to change (if possible) their processing time by choosing a shorter mode according to the negative minimum value of all the successors' requests. If the activity already has the mode with the shortest processing time, it sends as an improvement the value 0. If it does not use the shortest modes then it chooses one where the value $(Mode_{new} - Mode_{old})$ is close to the requested value. This difference will be sent as an improvement to all its successors. If the new value that the activities receive do not solve their delay problem, they should change their own modes too. They send to the successors the sum of their improvements and the best improvement of the predecessors. This continues until all requests are completed.

Although the steps, the communication and the rules are simple, it is impossible to know the real effect of the changes. The real influence of a new chosen mode according to the requested time on the model is not known in advance. The model is too complicated and has too many factors (unpredictable break/breakdown of resources, delay of activities cost by dispatching rules, etc.). If at least some activities have a start delay or a new processing time according to these factors, a new simulation will be required in order to see the full dimension of the effects. Besides that the executed activities are not always the same in such production plans (alternative paths).

Sometimes the system is moving around a local optimum and it cannot find more suitable scenarios. For that reason we created a special case. If in the last scenario, only *waiting for constraints* time slices were checked by the activities' decisions, without changing a mode or an earliest start date of any activity, then the activity can communicate additionally with the required resources. Only activities with exceeded due dates are allowed to communicate. By analyzing the resource state history the activity tries to find out, which activities allocated the required resources and lead to their start delay. The main reasons for this situation are:

- A low prioritized activity, despite of the existence of available resources at the beginning.
- Resource allocation for a longer period of time by other activities.

The activity has two possible requests to send:

- For an activity to start earlier than many of the high prioritized activities with the intention to start earlier (case 1).
- For all the activities to finish earlier (case 2).

The important aspect of these solutions is that their influence on the results can be substantial and that during the next steps the system will try to balance again.

3.2.2 Secondary Goal Function - Minimizing Slack and Balancing Resource Utilization

In case the model does not violate any due dates, the algorithm works on the secondary goal function, to optimize the resource utilization and to minimize the activities' slack. This is more complex than the first one. In the first goal function there is only one rule but in the second goal function, different rules try to optimize their own situation, while affecting each other negatively.

The first rule tries to minimize the *waiting for constraints* time slices further between the activities as in the primary goal function. The principle is the same as in Figure 4, with the only difference, that the activities try to start as early as possible according to their earliest start date (not according the violation time). The second rule tries to slow down the activities to minimize their slack, while requiring less resources. In general, non due date violation and availability of unused resources is an indicator to accept new orders, without changing the capacity of the facility or increasing costs.

Like in the first goal function in Figure 4, the algorithm works also backwards but the activity shares with its predecessors the difference between the adjusted earliest start date and the activity start time. The activity tries now to reach the theoretical maximum of the adjusted earliest start date. On the other hand in the first goal the difference is only used to avoid a due date violation. Every activity sends a request according only to its own value. If the activity does not have any *waiting for constraints* time slices then it does not send anything.

At the same time another communication strategy flows over the system. The activities that do not have any *waiting for constraints* time slice, can slow down their processing time. For that reason they first ask the successors if this action can violate its due date and inform the predecessors about the possibility to slow down their processing time. It is easy to observe that the active units of the system try to fill the gap in the activity time-line and reach a minimum slack at the same time. The difficulty on parallel goals lies in the fact that the activity cannot decide when different actions are requested. As a result we decided that an activity will not optimize its state if requests of both rules exist. After all decisions have been made the two rules control independent subgroups of the production plan.

In situations where the system is moving around a local optimum while the secondary goal function is in operation without having better results, we created a special case. If there are no changes to any activities' mode, earliest start dates and no due date violation exist at the last scenario, then the algorithm tries to change the earliest start date of activities that have not received any request. The ESD of activities will be changed only from activities that will not violate their LED and that of their successors. At the same time the predecessors can slow down their mode further. In most of the cases we can expect, that the new scenario will violate some due dates and as a result the algorithm will start again from the first step. This may not necessarily be a negative development as our purpose is also to balance the system.

4 EXPERIMENTS AND RESULTS

The experiments are based on two categories. Firstly we experiment on simple models (up to 1000 activities) with high resource availability and enough positive slack. The system found a scenario without due date violation after a small numbers of steps as well as a relatively good balance between the resource utilization and minimization of slack. On the second category we focus on complex models with very high potential of due date violation and low resource availability. The maximum number of steps was 50. In this case it was almost impossible to find scenarios without due date violation and the secondary function was not able to run for more steps because something was violated. So the balancing of the system proved difficult. An example of the activities violation of one of these models can be found in Figure 5.

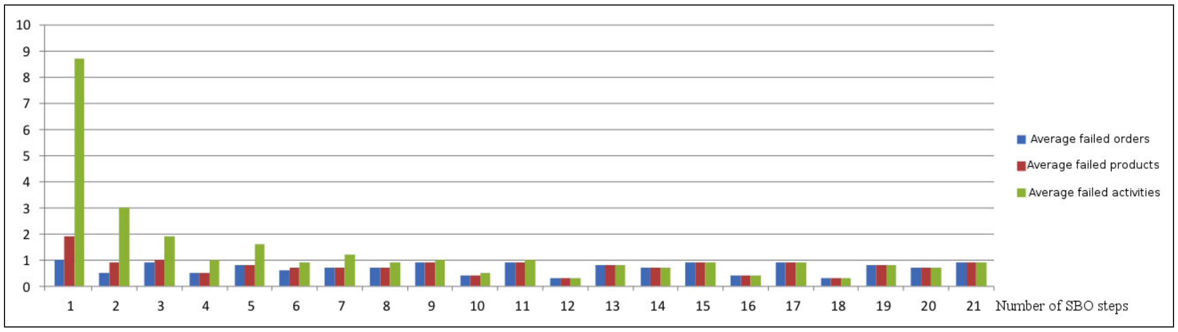

Figure 5: Results of a model with very high potential of due date violation and low resource availability.

5 CONCLUSION AND FUTURE WORK

The results have shown that the algorithm works and it is a suitable idea to use decentralized control to minimize the complexity. We are currently testing more complex models to evaluate the algorithm and obtain better results when the slack is relatively small. We try to take into account the list of the dispatching rules, so that the system could change them.

In addition we focus on alternative paths, which pose a lot of problems. We try to find common characteristics between selected and not selected paths in order to change the alternative path accordingly. Another part that we are working on at this point is to find a minimum size of the *waiting for constraints* time slice to avoid having to calculate it too regularly during optimization.

Lastly we created successfully the special cases for both goal functions overcoming constraints and obstacles in the realization process. Some activities had already sent their request to the predecessors when they received a request from activities out of their production plan. The activities had to resend new requests, which in turn slowed down the operation time of the algorithm and in some worst-case scenarios created deadlocks between activities that communicate with each other (circle communication).

ACKNOWLEDGEMENTS

We would like to thank Steffen Seifert for his implementation work and discussions during the project.

REFERENCES

10gen 2012. “mongoDB technology”. Accessed April. 15, 2012. http://www.mongodb.org.

Angelidis, E., A. Naumann, and O. Rose. 2012. “An Extended Critical Path Method for Complex Assembly Lines”. In *Proceedings of the 2012 Industrial Engineering Research Conference*, edited by G. Lim and J. W. Herrmann. Norcross, Georgia: IIE Institute of Industrial Engineers.

Angelidis, E., F. M. Pappert, and O. Rose. 2011, December. “A Prototype Simulation Tool for a Framework For Simulation-based Optimization of Assembly Lines”. In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 2383–2394. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

April, J., F. Glover, J. P. Kelly, and M. Laguna. 2003, December. “Simulation-Based Optimization”. In *Proceedings of the 2003 Winter Simulation Conference*, edited by S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 71–78. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Brucker, P., and S. Knust. 2006. *Complex scheduling*. Berlin ; Heidelberg [u.a.]: Springer.

Camazine, S., J. L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. 2003. *Self-organization in biological systems*. Princeton University Press.

Law, A. W., and M. G. McComas. 2002, December. “Simulation-Based Optimization”. In *Proceedings of the 2002 Winter Simulation Conference*, edited by E. Yücesan, C. H. Chen, J. L. Snowdon, and J. M. Charnes, 41–44. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Noack, D., and O. Rose. 2008, December. “A Simulation-Based Optimization Algorithm for Slack Reduction and Workforce Scheduling”. In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, 1989–1994. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

OSGi Alliance 2012. “OSGi technology”. Accessed April. 15, 2012. http://www.osgi.org.

Pappert, F. M., E. Angelidis, and O. Rose. 2010, December. “Framework for simulation based scheduling of assembly lines”. In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 1690–1968. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pinedo, M. 2007. *Planning and scheduling in manufacturing and services*. Corr. as of the 3. print. ed. New York, NY: Springer.

Pinedo, M. 2008. *Scheduling / theory, algorithms, and systems*. 3. ed. New York, NY: Springer.

Reuter, H., and B. Breckling. 1994. “Self organization of fish schools: an object-oriented model”. *Ecological Modelling* 75-76 (0): 147–159.

Seeley, T. D. 2002. “When Is Self-Organization Used in Biological Systems?”. *Biological Bulletin* 202:314–318.

Shapiro, A. 1996, December. “Simulation Based Optimization”. In *Proceedings of the 1996 Winter Simulation Conference*, edited by J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 332–336. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Steinberg, D., F. Budinsky, M. Paternostro, and E. Merks. 2008. *EMF: Eclipse Modeling Framework (2nd Edition)*. 2nd Revised ed. Boston: Addison-Wesley Professional.

T’kindt, V. 2006. *Multicriteria Scheduling : Theory, Models and Algorithms*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG.

AUTHOR BIOGRAPHIES

EVANGELOS ANGELIDIS is a Ph.D. student at University of the Federal Armed Forces Munich. He is a member of the scientific staff of Prof. Dr. Oliver Rose at the Chair of Modeling and Simulation. He received his M.S. degree in Computer Science from Dresden University of Technology. His research focuses on the simulation and optimization of complex assembly lines. His email address is evangelos.angelidis@unibw.de.

DANIEL BOHN is a Software Engineer at Bosch Software Innovations. He is also a Ph.D. student at University of the Federal Armed Forces Munich. He is a member of the scientific staff of Prof. Dr. Oliver Rose at the Chair of Modeling and Simulation. He received his M.S. degree in Computer Science from Dresden University of Technology. His research focuses on the simulation and optimization of complex assembly lines. His email address is daniel.bohn@unibw.de.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Department of Computer Science of the University of the Federal Armed Forces Munich, Germany. He received an M.S. degree in applied mathematics and a Ph.D. degree in computer science from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories. He is a member of IEEE, INFORMS Simulation Society, ASIM, and GI, and General Chair of WSC 2012. His email address is oliver.rose@unibw.de.