Simulation-based Deadlock Avoidance and Optimization in Bidirectional AGVS

Iryna Felko
Department of Computer Science
TU Dortmund
D-44221 Dortmund, Germany
iryna.felko@udo.edu

ABSTRACT

Autonomous guided vehicle systems (AGVSs) are popular in production systems, container ports, and intra-logistics. An efficient deadlock avoidance algorithm preventing long vehicle blockings and solutions to idle vehicles blocking others are required for bidirectional AGVSs. A symbiotic simulation resources scheduling decision support method is developed, including identification of route segments where deadlocks can potentially occur and an accordingly integrated banker's algorithm. Based on multiple online what-if simulations, the best deadlock safe resources schedule is determined for each short period just before the vehicles move. This proactive what-if analysis of resources utilization, alternative routes and dynamic parking strategies allows the minimization of the vehicles' total blocking time. Combining symbiotic simulation and a real time control of autonomous guided vehicles increases the AGVS's efficiency – in terms of deadlock safety and minimizing the total vehicle blocking time. This method enables AGVS performance evaluation under arbitrary dispatching, routing, scheduling and guidepath design strategies.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Types of Systems—*decision support, logistics*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*heuristic methods, scheduling*

Keywords

symbiotic simulation, autonomous guided vehicle, deadlock avoidance, DES, banker's algorithm, idle vehicle dynamic positioning

1. INTRODUCTION

Autonomous guided vehicles (AGVs) are advanced material handling devices used to transport pieces among the workstations in manufacturing facilities, warehouses and transshipment terminals ([15], [8]). Usually AGVS are directed by a central real time controller ([15]) operating AGVs and implementing control strategies for tactical decisions. Operating vehicles in such a system implies dealing with well-known highly complex traffic problems such as collision prevention, deadlock avoidance and scheduling. This paper focuses on deadlocks that are caused by guidepath sharing in a bidirectional AGVS (autonomous guided vehicle system). The related work on the field of deadlock avoidance in manufacturing systems has been done ([9], [19], [16]). Often the drawbacks of strategies that are used in practice remain the restrictivity which often leads to a decrease of resource utilization in a system. A deadlock avoidance strategy in bidirectional AGVSs taking the total blocking time of vehicles into consideration needs improvement.

In AGVSs resources are assigned to vehicles and released from vehicles after reaching the next checkpoint position on the guidepath, thus resulting in a sequential resources assignment problem specification. If the system contains shared resources, decisions to appropriately assign resources to vehicles can have serious effects on system performance, e.g. resulting in blocking as a temporary effect. A deadlock occurs when blocking develops into a circular waiting situation involving at least two vehicles ([6]). Then the activity of involved AGVs comes to a halt until it is resolved by external intervention. In this paper, the method for deadlock avoidance for an AGVS modelled as a sequencial resource allocation system (RAS) ([19]) is developed. The method considers the total vehicle blocking time caused by vehicle interference and waiting time as a side effect of deadlock avoidance, thus determining the resources schedule minimizing the total travel costs. Another important issue studied in this paper, which has been considered in the online simulation, is idle vehicle positioning ([11]). Most scheduling problems suppose that, vehicles can stay on their last job's destination position, however this is not true with some AGVS that do not have dedicated docking and parking locations for vehicles. We define the idle vehicle positioning problem for such systems and also integrate developed strategies into the online simulation ([10], [2], [1], [14]), which enables us to analyse their performance. AGVS is modelled as DES. The corresponding model contains the predefined set of vehicles' routes and is used in a deadlock avoidance algorithm. Of course, discrete event simulation of AGVs is not a new idea, but a combination of deadlock avoidance algorithms with an online simulation allows one to determine the best resource assignment strategy for each short period just before the vehicles move, thus supporting and optimizing an AGVS.

SIMUTOOLS 2011, March 21-25, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245542

The progress of developed what-if simulations is surely deterministic, but this assessment of the future can be used to detect future critical conditions and to compare the impact of choosing different alternative and parking routes on the total vehicle blocking time. Online simulation systems that interact and exchange information in real time with the physical system are still missing on the field of AGVSs. The rest of this paper is organized as follows. Section 1.1 describes the sequential resource allocation and greedy deadlock avoidance strategies existing in the reference system. Section 2 develops a deadlock avoidance strategy based on vehicle routes comparison. The AGVS model and optimization are described in section 3 along with an example. The integration of the banker's algorithm into the online simulation is discussed in section 4. A description of an idle vehicle problem and solutions to it are presented in section 5. Finally, the last section presents some conclusions drawn from our work and some prospects for further research issues in the field of AGVS.

1.1 Preliminaries

In our resources scheduling model the AGV's route is known in advance and is represented as a sequence of resource sets $p =< rs_1, rs_2, ..., rs_n >$. A resource set rs_i consists of a path $e(v_j, v_k)$ and its destination position v_k (both corresponding to an edge and a vertex in a guidepath graph ([19])). The resource set assigned to a vehicle corresponds to an atomic movement of the vehicle from one position to the next adjacent one. Before a vehicle movement is permitted the corresponding resources set has been requested and successfully granted by a scheduler instance. Initially a vehicle allocates some position v_1 and before moving requests a first resource set $rs_1 = \{e(v_1, v_2), v_2\}$ on the assigned route. Being granted the needed resources it moves to the next point v_2 along the path $e(v_1, v_2)$. Reaching the position v_2 it releases the allocated initial position v_1 and requests the next resources set $rs_2 = \{e(v_2, v_3), v_3\}$. The vehicle's operations order is relevant to the scheduling process. For further consideration the representation of the AGV's route is reduced to a sequence of positions $p =< v_1, v_2, ..., v_n >$.

The reference system ([18]) uses a *greedy* deadlock avoidance strategy based on dynamic zone planning ([15]). Dynamic zone planning is an efficient method to avoid deadlocks and collisions, where zones are not fixed and can be changed according to the traffic flow in the system. The main idea is, that when a vehicle arrives at a zone, the controller checks for the presence of another vehicle in this zone. If a vehicle is already traveling in this zone, then the vehicle intended to enter that zone has to wait until the other has passed. Thus, a requested resource set is only granted to a vehicle, if all resources of the route are available at once. This strategy is effective with regard to deadlock avoidance excluding all potential vehicle interferences but is inefficient with regard to resource utilization.

2. IDENTIFICATION OF CRITICAL PATH SECTIONS

We give a formal description for the emergence of deadlock if two AGVs move along the same path in the opposite direction. As described in the preliminaries, consider a given AGV route as a sequence of resource sets; such a route can be represented as a sequence of positions. For such a given route, all positions will be covered in the order defined by the sequence. E.g. the route for AGV V_1 is $p = (a_1, a_2, ..., a_n)$ and the route for AGV V_2 is $q = (b_1, b_2, ..., b_m)$. If two given AGVs move along the same path in the opposite direction, it holds that $a_i, ..., a_j = b_l, ..., b_k$, $b_l, ..., b_k \in rev(q)$ and $|a_i, ..., a_j| \geq 2$. If such similar segments in two resource sequences can be found, the corresponding path segments will be shared by both competing AGVs. To avoid a deadlock shared path segments can be occupied by only one vehicle at a time. Figure 1 illustrates a deadlock occurence for two vehicles V_1 and V_2 moving along the same path in the opposite direction. Examining the sequence p and the sequence $rev(q)$, we can find a correspondence between two routes, namely a shared subsequence of positions $a_i, ..., a_j = b_l, ..., b_k$ with a length ≥ 2 (at least two resources are required for forming a deadlock involving two AGVs). If we do find such a shared subsequence, the following sufficient condition may hold for the occurence of a deadlock involving V_1 and V_2: The position a_i has been allocated by V_1 and the next position in V_1's resources sequence is a_{i+1}. The position b_j has been allocated by V_2 and the next position in V_2's resources sequence is b_{j+1}. Furthermore $a_i = b_{j+1}$ and $b_j = a_{i+1}$.

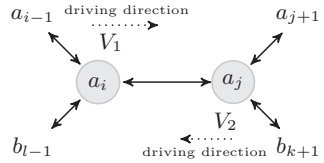

Figure 1: The deadlock with involving vehicles V_1 and V_2: V_1 allocates position a_i and requests a next position $a_j = b_k$, since V_2 allocates position a_j and requests a next position a_i. Both AGVs are blocked and must wait for resource release.

This situation represents a classical deadlock, no AGV can continue processing its assigned route. The requested resources a_{i+1} and b_{j+1} can only be granted if the currently allocated resources a_i, b_j are released by both AGVs, but no resources can be released because the requested resources are not available. Such deadlock situations would typically be resolved manually. We define a resource sequence $a_i, ..., a_j$ as a *critical section* belonging to V_1. Since $a_i, ..., a_j = b_l, ..., b_k$ and the *critical section* is a shared subsequence of resources that must be used in the given order, the *critical section* of AGV V_2 is defined as $b_k, ..., b_l$.

We developed an algorithm for determining critical path sections for all AGVs in a system. Such critical path sections represent a structural precondition for the deadlock occurence: If AGVs allocate resources within the same critical section simultaneously or if one AGV enters a critical section while the other one is already within, a deadlock will occur.

The length of a critical section is bounded below by 2. With a critical section of a length greater than 2 a deadlock would arise somewhere along the critical section if no of involved AGVs is not either rerouted or forced to wait outside the considered critical section. Representing an AGV route as a sequence of check points, we convert the two considered AGV routes into resource sequence notation $p = (a_1, a_2, ..., a_n)$ and $\tilde{q} = (b_m, b_{m-1}, ..., b_1)$, with the resource

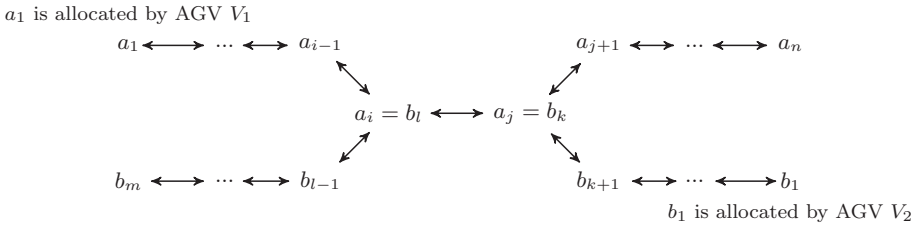

Figure 2: The relation between the path $p = (a_1, ..., a_{i-1}, a_i, a_j, a_{j+1}, ..., a_n)$ for AGV V_1 and the path $q = (b_1, ..., b_k, b_{k+1}, b_l, b_{l-1}, ..., b_m)$ for AGV V_2, where $\leftrightarrow$ denotes guide paths.

sequence of the second AGV being considered in reversed order.

We extend the above ideas to find all pairs of segments with the length bounded below by 2, one from each of two long sequences, such that there is no other pair of segments with greater homology. This problem corresponds to the optimal local alignment problem in molecular sequence analysis, so the Smith-Waterman algorithm can be applied to find all similar segments in two resources sequences ([21], [17]). The Smith-Waterman algorithm uses a dynamic programming method for generating the alignment matrix providing sequence alignment score values. Ordinarily, the Smith-Waterman algorithm computes an optimal local alignment for two given sequences. If we consider and follow all paths with a length ≥ 2 in the generated alignment matrix, all local alignments can be determined.

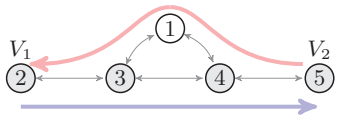

Figure 3: The part of transportation net with highlighted drive directions of AGVs. The AGV V_1 has to travel the route $A = (2, 3, 4, 5)$ and the AGV V_2 has to travel the route $B = (5, 4, 1, 3, 2)$, both represented as a sequences of check points in the transportation net.

Consider the example in figure 3 where two vehicles share the same subsequence of positions. The corresponding alignment matrix M computed by the Smith-Waterman algorithm on the sequences A and B is illustrated on figure 4.

$s[i, j]$ contains the score value and the appropriate arrow for the entry that has maximized this score. The value 2 in $s[2, 2]$ is the alignment score for the first similar subsequence $s_1 = (2, 3)$. The next value 2 in $s[5, 4]$ in the lower right corner of the table is the alignment score of a subsequence $s_2 = (4, 5)$ of A and $B' = rev(B)$. For $i, j > 0$, the value $s[i, j]$ depends only on whether $a_i = b_j$ and the values $s[i-1, j-1]$, $s[i-1, j]$ and $s[i, j-1]$, which are computed before $s[i, j]$. To reconstruct the elements of homologous subsequences, follow the arrows in $s[i, j]$ from the lower right corner. Each $\nwarrow$ on the path corresponds to an entry for which $a_i = b_j$ is a member of a local alignment.

Consider one part of the transportation net represented in the figure 2. The solution of the local optimal alignment problem delivers correlations for two resource sequences $p = (a_1, a_2, ..., a_{n-1}, a_n)$ and $\tilde{q} = (b_m, b_{m-1}, ..., b_2, b_1)$ resulting in similar subsequence $s_1 = (a_i, a_j = b_l, b_k)$. The result of

	j	0	1	2	3	4
i		b_j	②	③	④	⑤
0	a_i	0	0	0	0	0
			↖	↑	↑	↑
1	②	0	1	0	0	0
			↑ ↖			
2	③	0	0	2 ←	1 ←	0
			↑	↑		
3	①	0	0	1 ←	1 ←	0
			↑	↑	↑	
4	④	0	0	0	1 ←	0
			↑	↑	↑ ↖	
5	⑤	0	0	0	0	2

Figure 4: The alignment matrix M.

the comparison of the sequences p and $\tilde{q}$ are:

$$p = a_1 \ ... \ a_{i-1} \overbrace{a_i \ a_j}^{s_1} \ a_{j+1} \ ... \ a_n$$
$$q = b_m \ ... \ b_{l-1} \underbrace{b_l \ b_k}_{s_1} \ b_{k+1} \ ... \ b_1$$

As a result of the comparison all determined critical sections have to be distinguished and each considered resource sequence has to be separated in segments. Then the resource sequence consists of segments corresponding to the critical sections and such segments containing the rest of the resource sequence. The following example illustrates the sequence separated in segments:

$$p = [a_1 \ ... \ a_{i-1}] \overbrace{[a_i \ a_j]}^{s_1} [a_{j+1} \ ... \ a_n]$$
$$\tilde{q} = [b_m \ ... \ b_{l-1}] \underbrace{[b_l \ b_k]}_{s_1} [b_{k+1} \ ... \ b_1]$$

The segment $[a_i \ a_j]$ is the critical section determined for the first considered route p. The segment $[b_k \ b_l]$ is the critical section for the second route q. These segments are the result of sequence alignment and a useful indicator that the route segment $[a_i a_j = b_l b_k]$ will be shared by both AGVs. If both considered AGVs allocate resources within the same critical section s_1 and request the resources in the given order a deadlock will occur. These insights have been used to develop an adequate deadlock avoidance strategy. Hence the deadlock avoidance policy may ensure mutual exclusion for

Algorithm 1 Preprocessing procedure

Require: Not empty set R with cardinality $n \geq 1$ containing resources sequences
Ensure: All local alignments for all n resources sequences

for all R_i such that $0 \leq i \leq n-1$ **do**
  for all R_j such that $0 \leq j \leq n-1 \wedge i \neq j \wedge R_i$ and $R_j$has not been compared **do**
    $R'_j \leftarrow reverse(R_j)$
    $A \leftarrow smith\_waterman(R_i, R'_j)$ {Smith-Waterman procedure returns a set A of all local alignments}
    if $A \neq \emptyset$ **then**
      $\tilde{R}_i, \tilde{R}_j \leftarrow$ split resources sequences R_i and R_j according to identified critical sections
      Create mapping for R_i to tupel $(R_j, \tilde{R}_i)$ and for R_j to tupel $(R_i, \tilde{R}_j)$ {The tupel $(R_j, \tilde{R}_i)$ consists of the compared resources sequence R_j and the corresponding splitted resources sequence $\tilde{R}_i$}
    end if
  end for
end for

resource utilization within the critical section. Before granting V_1's resources request, the policy checks if a competing AGV V_2 has already allocated resources inside this critical section. If so, V_1's resource request will be rejected, which results in V_1 being blocked as long as V_2 holds resources in the critical section. In the case that both competing AGVs want enter the critical section at the same time, the decision maker instance has to determine a schedule for participating AGVs, i.e. the order of resource allocation. This decision impacts the global blocking time in the system either negatively or positively. Hence the problem under study is not only deadlock avoidance during real-time control but also a simulation-based scheduling mechanism. In our proposed framework we demostrate a simulation predicting future system performance based on current decisions on resource allocation by running in fast mode. The results of the fast mode analysis will be used by the decision maker instance in control mode to avoid deadlocks and to minimize total blocking times in the system.

The deadlock avoidance preprocessing procedure (see algorithm 1) perfoms a resource sequence comparison for routes of all active AGVs in the system. It is possible that a resource sequence contains shared subsequences with more than one other resource sequence in more than one local alignment segment. Then the information about resources sequences is stored in a predefined data structure containing a mapping of a considered resource sequence r_i to compared routes $r_1, ..., r_z$ and determined critical sections for all compared routes. Once the data structure has been constructed, the information about critical sections and corresponding routes can be efficiently acquired at runtime. The Smith-Waterman algorithm runs in $O(c^2)$, c being the average route length for the given transportation net. The preprocessing procedure must be performed for all n routes, resulting in a runtime complexity of $O(n^2c^2)$. When a new route is added to the system, the corresponding resource sequence is compared with resources sequences of all existing routes.

3. ONLINE SIMULATION METHOD

The main goal of the developed online simulation method is to determine the system performance in terms of global blocking and processing times, which is a primary performance measure that can be negatively affected in case of a deadlock in AGVS scheduling. If a deadlock occurs, the total blocking time t_b increases until the deadlock is resolved either automatically or manually. Thus, the simulation-based scheduling mechanism for resources assignment is developed using online simulation as a look-ahead scheduling and deadlock avoidance module. In the online AGV simulation, we use our model of physical state resources allocations by vehicles including all operating and idle AGVs. This model of a physical system is our picture of reality and is used in "what-if" simulations ([1], [2], [14]) to determine how resources assignment to vehicles should be to process as effectively as possible.

Assume that two AGVs V_i and V_j compete for resources within the same critical section. Two situations are then possible. In the first case, the AGV V_i has reached the critical path section first and already allocates the required resources inside the critical section. In the second case, two AGVs have reached the critical path section at the same time and request the needed resources concurrently.

Taking information about the deadlock threat into account, the scheduler instance has to make a decision regarding the resources assignment and permit the one of two competing vehicles V_j to enter the critical path section. This decision implies that one of the competing vehicles has to wait until all resources within the critical section have been released by V_i.

Each time competing AGVs are identified, i.e. resources within a critical section are requested, a set of "what-if" simulations is started. A "what-if" simulation is initialized with a potential future system state, as it would be if the chosen vehicle has obtained the required resources. The number of "what-if" simulations started is the same as the number of competing AGVs for the considered critical path section. A single simulation runs until it has reached a state in which all AGVs have completed their assigned jobs. If a resource assignment leads to a deadlock this "what-if" simulation can be terminated. We compare the results of different resources assignments in the same simulated future and evaluate the impact of subsequent scheduling decisions on the total blocking time t_b, i.e. the deadlock occurence rate. Comparing these simulation results determines the best deadlock safe execution order for the given set of routes. This best execution order is applied directly to the physical system; the resulting system state is then fed back to the simulation, which makes the simulation a symbiotic one ([1]).

Given the current allocations of all vehicles, the allocations and corresponding vehicle movements for the next step are computed. In each "what-if" simulation resource alloca-

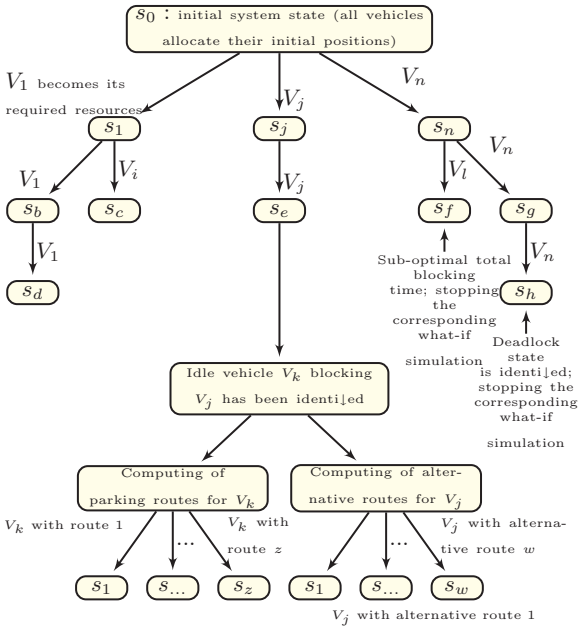

Figure 5: Exploration of a system state space.

tions of a system for time t and the effect of choosing a vehicle being assigned its required resources within the critical section are estimated. All possible vehicle movements can be evaluated via a one-step simulation procedure to determine whether it results in a deadlock and to determine the corresponding total blocking time. Given a set of resources will be assigned to all active vehicles in the system in the next time step, the algorithm considers the system state if all movements are made. In our model we assume that there are no AGV failures and that processing times and AGV speed are deterministic due to the nature of the autonomous system. During one simulation step, each active AGV being assigned its needed resources may act. The results of these actions are updated synchronously and the simulation advances.

The system state is modeled as follows: it contains the number of AGVs operating in the system. For each vehicle the following information is stored: the currently allocated resources; the whole vehicle's route, i.e. complete sequence of required resources; the requested resources set, i.e. resources required for the next vehicle move; the total blocking and driving time; the boolean flag, if the vehicle is forced to wait. Note, that only active resources requestors (vehicles) and existing critical sections/other shared resources have an influence on the decisions within a what-if simulation. Then a system state space is explored as shown in Fig. 5. $V_1, ..., V_n$ being competing vehicles requesting resources within the same critical section and/or other shared resources, i.e. competing for the same path/point. All possible resources asignments are considered and results in different system states $s_0, ..., s_w$. The transition from one state to another one corresponds to the simulation of vehicles' moves until conflicts occur. The state s_d is the termination state, i.e. all jobs are completed and evaluation of total blocking time can be performed.

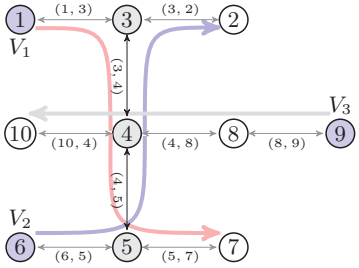

Figure 6: The initial physical system state and routes to be traveled by vehicles V_1, V_2 and V_3 (The assigned positions are marked with a blue background).

3.1 Example

The following example shows how distinct what-if simulations (WIS) are initialized and executed. There are three active vehicles V_1, V_2 and V_3 in the AGV system. The corresponding guidepath graph and vehicles' initial positions are illustrated in Fig. 6. Their routes can be represented as sequences of resources sets (as explained in section 1.1).

The route of V_1 is $R_{V_1} = (\{\mathbf{1}\}, \{(1,3),\mathbf{3}\}, \{(3,4),\mathbf{4}\}, \{(4,5),\mathbf{5}\}, \{(5,7),\mathbf{7}\})$, the route of V_2 is $R_{V_2} = (\{\mathbf{6}\}, \{(6,5),\mathbf{5}\}, \{(5,4),\mathbf{4}\}, \{(4,3),\mathbf{3}\}, \{(3,2),\mathbf{2}\})$ and the route of V_3 is $R_{V_3} = (\{\mathbf{9}\}, \{(8,9),\mathbf{8}\}, \{(4,8),\mathbf{4}\}, \{(10,4),\mathbf{10}\})$. These sequences of resources sets can be represented as sequences of guidepath positions that may be passed through by vehicles in the predefined order:

- $R_{V_1} = (1, 3, 4, 5, 7)$,
- $R_{V_2} = (6, 5, 4, 3, 2)$,
- $R_{V_3} = (9, 8, 4, 10)$.

These resources sequences containing only positions will be compared with each other as described in algorithm 1. For the first two routes R_{V_1} and R'_{V_2} we obtain one homologeous subsequence $s_1 = (3, 4, 5)$:

$$R_{V_1} = 1 \overbrace{3\ 4\ 5}^{s_1} 7$$
$$R'_{V_2} = 2 \underbrace{3\ 4\ 5}_{s_1} 6$$

Exploring the information about the detected critical section s_1 the first two routes can be separated into regions corresponding to the critical section and to the rest of the route. The separation of V_1's route yields

$$[\{1\}], \overbrace{[\{(1,3),3\}, \{(3,4),4\}, \{(4,5),5\}]}^{s_1}, [\{(5,7),7\}],$$

since positions $3, 4, 5$ are contained in s_1. For V_2's route, the result is

$$[\{6\}], \overbrace{[\{(6,5),5\}, \{(5,4),4\}, \{(4,3),3\}]}^{s_1}, [\{(3,2),2\}].$$

V_3's route does not share critical sections with other routes, so it consists of only one segment $R_{V_3} = [\{9\}, \{(8,9),8\}, \{(4,8),4\}, \{(10,4),10\}]$.

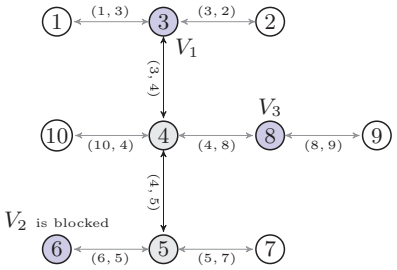

Figure 7: The first what-if scenario, in which V_1 is being assigned its needed resources $\{(1,3),\ 3\}$.

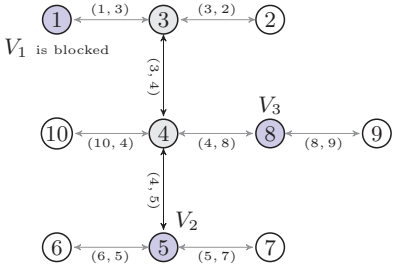

Figure 8: The second what-if scenario, in which V_2 is being assigned its needed resources $\{(6,5),\ 5\}$.

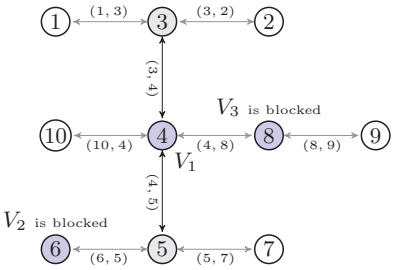

Figure 9: The what-if simulation, in which V_1 is being assigned its requested resources $\{(3,4),\ 4\}$.

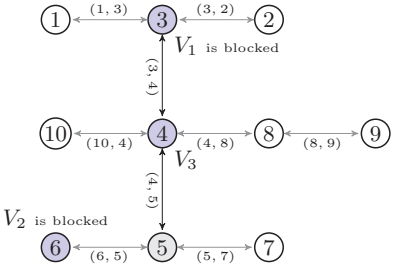

Figure 10: The what-if simulation, in which V_3 is being assigned its needed resources $\{(8,4),4\}$.

Then the system state corresponding to the real system state as shown in Fig. 6 is created. It displays current resources allocations. As the simulation advances, the next vehicles' requests of required resources are proceed. V_1 needs the resource set $Req_{V_1} = \{(1,3),3\}$, V_2 needs $Req_{V_2} = \{(6,5),5\}$ and vehicle V_3 requests $Req_{V_3} = \{(8,9),8\}$. Additionally, V_1 allocates resource 1, whereas V_2 and V_3 allocate resource sets with only one position, 6 and 9, respectively. Both resources requests of vehicles V_1 and V_2 are contained in the identified critical section s_1, precisely $Req_{V_1} = \{(1,3),3\} \in [\{(1,3),3\},\ \{(3,4),4\},\ \{(4,5),5\}]$ and $Req_{V_2} = \{(6,5),5\} \in [\{(6,5),5\},\ \{(5,4),4\},\ \{(4,3),3\}]$. Both AGVs want to enter the critical section concurrently and the scheduler instance must reject the resources request for at least one AGV. Otherwise a deadlock involving AGVs V_1 and V_2 within the critical section (3, 4, 5) will occur. Only one vehicle may enter the critical section at a time. So one vehicle will be permitted to enter the critical section and another one will be forced to wait. Two what-if simulations are initialized with the described current system state in order to find out which vehicle may be chosen to enter the critical section.

The first what-if simulation is run with V_1 chosen for receiving its needed resources, where V_2 is blocked and must wait. The resulting situation is shown in Fig. 7.

The second what-if simulation is run with V_2 being assigned its resources and V_1 being blocked until V_2 releases all resources within the critical section s_1. The resulting configuration is shown in Fig. 8.

We now consider the first what-if simulation, where the requested resources have been assigned to vehicles V_1 and V_3. Suppose both vehicles need the same time Δt to travel along paths $(1,3)$ and $(8,9)$. At the next simulation step the next resources sets in the sequences are requested. These are $Req_{V_1} = \{(3,4),\ 4\}$ and $Req_{V_3} = \{(4,8),\ 4\}$, since V_2 is still blocked outside of the critical section.
As $\{(3,4),\ 4\} \cap \{(4,8),\ 4\} \neq \emptyset$, there is a simple competition for position 4. In this case the next set of what-if simulations is initialized to find out which vehicle should be chosen to pass through first.

If the requested resources set $Req_{V_1} = \{(3,4),\ 4\}$ is assigned to V_1, the vehicle V_3 is blocked, too (see Fig. 9). It waits until V_1 reaches position 5 and then releases the resource 4. Thus the blocking time for V_2 increases to $t_b(V_2) = 2$ (if we assume the costs for traveling the path are 1 and this assumption holds for all paths in the considered guideline graph: costs for traveling the path $(3,4)$ plus costs for traveling the path $(4,5)$. At the same time V_2's blocking time is bounded by $t_b(V_2) = 4$. When the what-if simulation terminates, i.e. all routes have been processed, the total amount of blocking time for all vehicles is $t_b = 2 + 4 = 6$.

The next what-if simulation delivers the worst resulting blocking times: The requested resources $Req_{V_3} = \{(4,8),4\}$ are assigned to V_3 and V_1 is forced to wait (see Fig. 10). It is blocked until V_3 reaches position 10 and releases the needed resource 4. This execution order of vehicles forces V_2 to wait longer for the needed resources, as then V_1 is still inside the critical section and is blocked, too. The blocking time is $t_b(V_1) = 2$ and the blocking time of V_2 increases to $t_b(V_2) = 6$ until V_1 reaches position 7, thus resulting in a total blocking time for all vehicles $t_b = 2 + 6 = 8$. In our simple example the execution order of the vehicles according to the simulation scenario shown in Fig. 9 will be reported for further decision support of the central controller. The precomputed optimal vehicles schedule can then be applied in a real time. The vehicle best suited for assignment of the needed resources will be propagated online by the decision support instance. For more complicated instances also

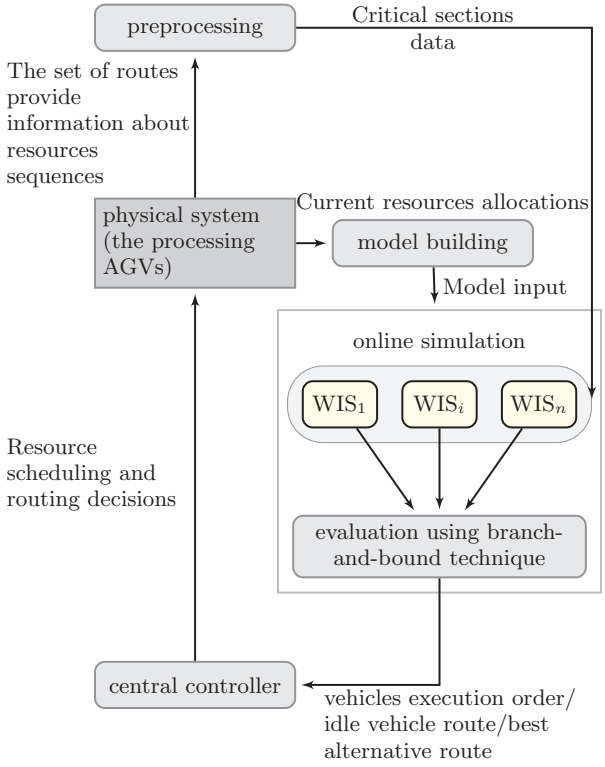

Figure 11: Symbiotic simulation system architecture

alternative routes will be considered (see section 5).

3.2 Optimization using branch-and-bound technique

The online simulation method is responsible for generation and simulation of different what-if scenarios (see Fig. 11). A what-if scenario is initialized with an image of a real system state. Only different configurations of resources assignments during the simulation distinguish a what-if scenario from another one. After the results of several correspective what-if simulations are obtained they can be evaluated and then recommended to the controller before a decision can be made in real time.

We have applied a branch-and-bound optimization technique for operating the reactive simulation run and for finding the optimal resource allocation schedule. Each simulation run investigates the performance of a different scenario of the resource allocation decisions between competing vehicles. The objective function is the sum of total blocking times of vehicles in the system for a given set of transportation tasks. If a deadlock occurs it holds $\sum_{i=1}^{n} tb_i = \infty$, n being the number of vehicles operating in the system. The goal is to find a deadlock safe resources schedule minimizing the sum of total blocking times. While waiting times are unavoidable, additional blockings can occur involving idle vehicles if no reparking or rerouting is done. Different execution orders of vehicles on critical sections and junctions, and also developed reparking strategies (see section 5), all representing a valid solution space, lead to different performance measurements resulting from the corresponding what-if simulations.

Using the developed deadlock avoidance policy, the infinite blocking time of deadlocked vehicles can be avoided, but waiting times for involved vehicles must be enforced. Then the value of the objective function $\sum_{j=1}^{k} tb_i$ results in a what-if simulation, $j \in E$ and E being the set of vehicles forced for wait.

Using a branching technique resulting from the exclusive resource allocation within the same critical section, the solution space can be divided. In our case for each possible and valid solution a single what-if simulation will be initialized. The vehicle being assigned its needed resources is chosen and the execution order of AGVs during the simulation is determined. This way decisions on resource assignment are made during the simulation. The solution space can be represented as a binary decision tree, where resources requests of AGVs are granted or not. The solution determined by the greedy heuristic (described in section 1.1) can be computed efficiently in polynomial time. Then the upper bound U for the objective function value is known, its value being U^+. The lower bound L is the best known solution. If resources schedule contains no blocking times its corresponding value L^+ is equal to 0.

When a what-if simulation results in a deadlock as an effect of the prior decisions on resources assignments, the objective value is equal to infinity. In this case the investigation of the corresponding subproblem can be aborted and the simulation can be terminated. Otherwise the optimum can potentially be found within the considered what-if simulation WIS_i. Then the upper bound value L_i can be determined. If $L_i > U^+$, the optimal solution cannot be found within the considered what-if simulation. Then this simulation can be terminated not yielding the optimal vehicle schedule. If $L_i \leq U^+$, the execution order minimizing total vehicles blocking time can potentially be identified in further simulation processing. In our case all initialized what-if simulations proceed in *first-in-first-out* order.

When an idle vehicle blocking others is identified in the system, a set of reactive what-if simulations are initialized. Then different solution strategies (see sec. 5) are gone through in corresponding what-if simulations. The value of the objective function can be determined for each selected dynamic positioning strategy. A solution space contains alternatives in second-best routes and idle vehicle positioning routes influencing directly the vehicles' total blocking time in a usual manner (further critical sections, possible deadlock formations, occurence of further blockings caused by idle vehicles). All these possible developments in the future and the consequences of decisions made for the current WIS can be evaluated using a branch-and-bound technique for determing if the current WIS could lead to the best execution order of AGVs or should be terminated.

4. INTEGRATION OF A BANKER'S ALGORITHM

We describe a variant of the banker's algorithm ([6], [13], [19]) for deadlock avoidance in AGVS and its integration into online simulation. In the banker's algorithm, processes

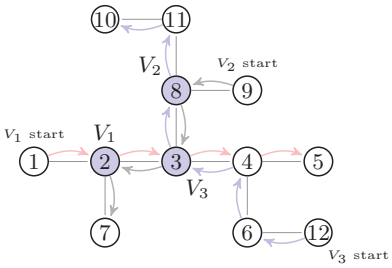

Figure 12: A deadlock has occured: each AGV needs one more resource than currently available before allocated resources can be released.

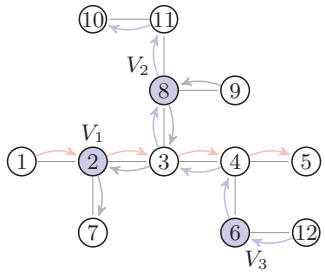

Figure 13: Deadlock avoided with a banker's algorithm.

are required to declare their maximum claims of resources in advance. The information about these maximum resource claims corresponding to the vehicle routes is normally available as AGV routes have been computed in advance. When a new request of resources needed by an AGV is made, the banker's algorithm would grant the request if the resulting system remains in a safe state, i.e. even in the worst case that all processes (AGVs) request their maximum claims, there is still a schedule of vehicles allowing all requests to be granted. Applying the original banker's algorithm in AGVSs, deadlocks can be avoided but restrictions can occur. It is possible that a request is denied because it is putting the system into an unsafe state, even though the system is deadlock-free. The banker's algorithm only knows the maximum claims from each process, thus causing situations known in the literature as restricted deadlocks making the algorithm inefficient in resource utilization. If we knew that an AGV never needs all resource sets in the sequence simultaneously and which critical sections exist, the original banker's algorithm can be efficiently integrated into online deadlock avoidance strategies.

In a what-if simulation the deadlock avoidance can be extended using the original banker's algorithm to guarantee the deadlock safety of the AGV's execution order recommended to the physical system. With the information about currently allocated resources and additionally required resources before the current allocated resources can be released, a banker's algorithm may be applied to determine whether granting a resource request to a vehicle is safe. The original banker's algorithm can be modified knowing the existing critical sections as opposed to the complete resources sequences for each AGV. A given set of resource requests is checked for resources contained in the same critical section. If there are any, the corresponding vehicles are identified and at least two what-if simulations are initialized. Then a vehicle forced to wait and a vehicle permitted to enter the critical section are known. The set of resource requestors is reduced by one waiting vehicle. If no reduction would be performed the original banker's algorithm would declare the resulting system state as unsafe and resources requests would not be granted. Thus, the only reduced set of resource requestors is considered when testing system safety. Additionally, the *rank* vector (i.e. the vector of maximum needed resources) for each AGV is reduced to the current route segment, either corresponding to the critical section or to the rest of a route that contains the requested resources. For the reduced set of resource requestors the system state safety condition must hold only for the route segment containing already requested resource, since an AGV rarely needs all resources belonging to the route simultaneously. In the online simulation, the resource requests will be granted if the resulting state is safe by applying the banker's algorithm considering only vehicles chosen to request their needed resources. The corresponding what-if simulation proceeds only if the resulting state is safe and generates one feasible solution which can be compared with possible another solutions. Deadlock states could potentially be detected within a set of what-if simulations without checking a state safety before. Applying a banker's algorithm provides additional safety and reduces a considered state space, because only safe states are generated and considered in further what-if simulation processing. We can only avoid deadlocks if the assumed system state is safe for considered execution order of vehicles, i.e. all future states form the given one are safe.

We introduce an example for deadlock avoiding using a banker's algorithm. Suppose three vehicles V_1, V_2 and V_3 sharing a critical section with each other (see Fig. 12). As soon as vehicle V_1 requests position 3 and V_3 requests the position 4 two what-if simulations corresponding to the concurrent resource users V_1 and V_3 will be initialized. We show that regardless of mutual exclusion on critical path section $[3, 4]$ complicated deadlock situations involving more than two AGVs can still occur. Consider in the first what-if simulation the request of V_3 to be granted; V_3 has entered the critical section $[4, 3]$ and V_1 is waiting at position 2 for passage of V_3. V_2 has requested the resource 3. Since position 2 is allocated by V_1, this request cannot be granted; then V_1 has already entered the critical section $[2, 3]$ shared with the requesting vehicle V_2. Therefore V_2 is blocked at position 8 obstructing V_3 having right of way and requesting position 8 in the next step. The circular wait condition is satisfied. Thus a deadlock has occured regardles of the deadlock avoiding policy considering only critical sections being effective for only two involved AGVs sharing critical section. The banker's algorithm with reduced maximum claims can be used within a what-if simulation.While the vehicle V_1 is constrained for entering the critical section $[3, 4]$ and must wait at position 3 for V_3 passing through, V_2 is still blocked too. When V_3 requests its needed resources. Then the system safety test is performed with the following data: $Req_{V_3} = \{4\}$ and $rank(V_3) = \{4, 3\}$, $rank(V_1) = \{3, 4\}$, $rank(V_2) = \{8, 3\}$. For $rank(V_3) = \{4, 3\}$ all resources are available in the system. But granting V_3's request, the banker's algorithm identifies the resulting system state is unsafe: V_3 would allocate position 3 and release position

4. No execution order including all vehicles could be determined obtaining the resources claims $rank(V_1) = \{3, 4\}$ and $rank(V_2) = \{8, 3\}$, these could not be completed. Using this information, the first what-if simulation can be identified as resulting in a deadlock if V_3 enters the critical section $[4, 3]$ and receives right of way before V_1 (see Fig. 13). This WIS can be terminated and the corresponding execution order of vehicles is reported to the central controller as one leading to a deadlock.

5. DYNAMIC PARKING STRATEGIES

Vehicle idleness is unavoidable in autonomous guided vehicle systems ([11]). If no vehicle depot or parking locations are defined in the transportation network, idle vehicles may be distributed over the transportation network. In this case idle vehicles could block other vehicles whose route contains their current parking position.

We develop strategies to resolve blockade situations involving an idle vehicle and based on the work of [5], [20], [3], [12], [7] and [4]. Several assumptions have been made in our framework: when a vehicle becomes idle, it allocates some dwell point until a transport order is received. In our framework we develop strategies for (a) disposing idle vehicles and (b) determing home locations for vehicles. The objective of these strategies is to minimize the total blocking time in the system. Both strategies are integrated into the online simulation. If a blockade involving an idle vehicle occurs, what-if simulations with two developed blockade resolving strategies are initialized. The results of these what-if simulations are analyzed and the strategy concluding minimum blocking times can be used for decision support by the real-time controller. The decision maker instance then proposes to the physical system either to dispose the idle vehicle or proposes an alternative route for the blocked vehicle. We define the idle vehicle blocking problem:
Suppose that an idle vehicle allocating some dwell point is given. Assume that no special parking positions are given in the system and the current transport orders pool is empty (i.e. new transportation tasks cannot be assigned to the idle vehicle). The vehicles that have finished the assigned transportation jobs will not be sent to predefined parking positions. Then an idle vehicle can become an obstacle for other operating vehicles in the system. These would be blocked by the idle vehicle V_i. Formally it holds for the idle vehicle V_i:

1. V_i allocates the dwell-point p_k being the final position in V_i's last travelled route,
2. $\exists j,\ j \neq i,\ i,\ j \in \{1, ..., n\}$ and n is the number of AGVs in the system. If it holds $p_k \in route(V_j)$ and the position p_k has not been covered by V_j, yet, blocking will occur. We denote V_j as a potentially blocked vehicle.

Then it is known in the system that vehicle V_j will be blocked requesting the position p_k. This blocking case can be prevented in advance if V_i can be disposed or an alternative route for V_j can be determined. The effectiveness of both strategies for minimizing the total blocking time is evaluated using an online-simulation method. If an idle blocking others is identified or predicted in the system the online simulation is triggered to try both the displacement route for the idle vehicle and the alternative route for the blocked vehicle for performance comparison. Then a decision can be made on the basis of resulting total blocking time and the best route can be suggested for the physical system fast enough to avoid idle vehicle blocking.

5.1 Idle vehicle displacement routing

We propose a polynomial-time shortest-deviation-path search algorithm to find a new displacement route for an idle vehicle V_i based on [12], [4]. Consider the route of a potentially blocked vehicle V_j that is given and can be represented a sequence of positions $route(V_j) = \langle (v_1 = v_s), (v_2), ..., (v_t) \rangle$, where v_s ist the start position node and v_t is the destination position node in the route. Furthermore it holds for the idle vehicle V_i that its currently allocated position $v_h \in route(V_j)$. Hence V_j's route can be split at v_h in two subroutes $w_1 = \langle (v_1 = v_s), (v_2), ..., (v_{h-1}) \rangle$ and $w_2 = \langle (v_h), ..., (v_t) \rangle$. Assume that the potentially blocked vehicle V_j moves along the first subroute w_1; once v_{h-1} is reached V_j becomes blocked. So the shortest deviation from the subroute $w_2 = \langle (v_h), ..., (v_t) \rangle$ including a new dwell point v'_h has to be determined. For each route point $v_i \in w_2$ each edge $e_j = (v_i, u)$, $u \notin w_2$ is considered. The shortest deviation path leading from the origin position v_h to the new destination position $u = v'_h$ is computed. The deviation path costs are defined as $c(w) = min\{c(v_h, v_i) + c(v_i, u) | v_i \in w_2, u \notin w_2\}$. For all n determined deviation paths the shortest path is selected as $w' = min\{c(w_1), ..., c(w_n)\}$ and then assigned to vehicle V_i. When computing the shortest deviation path the lower bound for path cost can be determined from already computed paths. If some path with greater costs than the upper bound occurs the computation for this path can be aborted. In the next iteration the following deviation point $v_i \in w_2$ can be selected for computation. An additional condition may hold for the new dwell point v'_h that is necessary to prevent further idle vehicles blocking others: v'_h is not already a dwell point for some other idle vehicle and v'_h is not a position that other vehicles still have to pass.

5.2 Alternative routes for blocked vehicles

Alternatively, redirection routes for potentially blocked vehicle can be determined ([4], [12]). First we define a set of forbidden edges as $P = \langle e_1, ..., e_m \rangle$ containing all $e_i \in E^+_{v_h} \cup E^-_{v_h}$, v_h is a dwell position of the idle vehicle V_i. For each edge $e_i = (v_j, v_k) \in P$ holds that $v_j = v_h$ or $v_k = v_h$. All input and output edges of the node v_h in the given transportation net are defined as forbidden edges and all redirection routes may not contain $e_i \in P$ avoiding request of position v_h. In our method we use the Hoffman-and-Pavley algorithm ([12]) to determine the best alternative route not containing forbidden paths. The algorithm computes the k-shortest path as redirection from the $k-1$-shortest path. All redirections for some path $p = \langle v_s, ..., v_t \rangle$ can be obtained by determining for each node $v_i \in p$ some edge $e = (v_i, u)$, $u \neq v_{i+1}$ and the path from this redirection node u to v_t. So the alternative route is composed of the path from the origin position v_s to the redirection node u and the shortest path from node u to destination node v_t. Formally it holds for the redirection path $q = \langle (u_0), ..., (u_m = v_t) \rangle$ of V_j's original route $w = \langle (v_1 = v_s), ..., (v_t) \rangle$:

- $\exists x,\ x \in \{1, ..., t\}$ and $x < m \wedge x < t$, such that $v_i = u_i$ with $0 \leq i \leq x$. The node sequences in two paths are identical up to the index x;

- There is some redirection node $u_{x+1} \neq v_{x+1}$ from the original route w;

- For each edge e_i in the redirection path q holds that $e_i \notin P$. And finally $\langle (u_{x+1}), ..., (u_m = v_t) \rangle$ is the shortest path from the u_{x+1} to v_t.

From the set of all computed redirection routes the shortest one is selected and assigned to the potentially blocked vehicle V_j.

6. CONCLUSIONS

This paper introduces the application of symbiotic simulation to avoid deadlocks and to minimize the total vehicle blocking time caused by vehicle interference that arises in an AGVS. The algorithm for determining critical sections according to the given routes set is developed. The algorithm identifies critical sections, i.e. paths that may not be travelled by vehicles in opposite directions at the same time. We use a critical sections approach for developing the deadlock avoidance policy; the vehicles are permitted to enter the critical section successively. All possible execution orders of vehicles could potentially be involved in a deadlock and competing vehicles allow us to initialize a set of what-if simulations and provide us with a mechanism to simulate the resulting system state.

The AGVS is modelled as a discrete event system. According to possible resource assignment decisions, a set of what-if scenarios can be generated. Using such models and taking into account information about critical sections, possible competing vehicle interferences and alternative/parking routes added to the system, the vehicles execution orders are determined to make decisions about resources assignments and vehicle movements in real time on the basis of the knowledge about the resulting system performance in the future. Comparing the results of multiple what-if simulations, minimization of the total vehicle blocking time can be achieved. Then the corresponding schedule will be recommended to the physical system. The simulation is used for studying the impact of choices about resources assignments to the vehicles on the total blocking times of the AGVs under the developed deadlock avoidance policy.

The main advantage of the developed online simulation method is that it helps studying the performance of an AGVS running in parallel with it and that decision impacts can be evaluated via simulation before the real system state changes avoiding possible bad resource assignments and routing decisions. Further enhancement of the developed deadlock avoidance strategy is forecasting of cyclic deadlocks by vehicles' routes analysis. If computational resources for running online what-if simulations in parallel with a real system are available, resource schedules and different routing scenarios can be computed and evaluated in advance. Future work will include dynamic updating of the model and further investigations of the methodology of the complex adaptive symbiotic simulation and their application to AGVSs.

7. REFERENCES

[1] H. Aydt, S. J. Turner, W. Cai, and M. Y.-H. Low. Research issues in symbiotic simulation. In *Winter Simulation Conference*, pages 1213–1222, 2009.

[2] H. Aydt, S. J. Turner, W. Cai, M. Y. H. Low, P. Lendermann, B. P. Gan, and R. Ayani. Preventive what-if analysis in symbiotic simulation. In *WSC '08: Proceedings of the 40th Conference on Winter Simulation*, pages 750–758. Winter Simulation Conference, 2008.

[3] R. Bellmann. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[4] J. A. de Azevedo, J. J. E. R. S. Madeira, E. Q. V. Martins, and F. M. A. Pires. A shortest paths ranking algorithm. 1990.

[5] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] E. W. Dijkstra. Cooperating sequential processes, technical report ewd-123. Technical report, 1965.

[7] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28:652–673, 1998.

[8] M. P. Fanti. Event-based controller to avoid deadlock and collisions in zone-control agvs. *Int. J. Prod. Res.*, 40(6):1453–1478, 2002.

[9] M. P. Fanti, B. Mmaione, S. Mascolo, and B. Turchiano. Performance of deadlock avoidance algorithmis in flexible manufacturing systems. *Journal of Manufacturing Systems*, 15(3):164–178, 1996.

[10] R. Fujimoto, D. Lunceford, and A. L. Uhrmacher. Grand challenges for modelling and simulation: Dagstuhl report. Technical report 350, Schloss Dagstuhl. Seminar No 02351, 2002.

[11] A. J. R. M. Gademann and S. L. van de Velde. Positioning automated guided vehicles in a loop layout. *European Journal of Operational Research*, 127(3):565–573, December 2000.

[12] P. Hoffman. A method for the solution of the nth best path problem. *Journal of the ACM*, 6:506–514, 1959.

[13] R. C. Holt. Some deadlock properties of computer systems. *ACM Comput. Surv.*, 4(3):179–196, 1972.

[14] F. Kamrani and R. Ayani. Simulation-aided path planning of uav. In *Winter Simulation Conference*, pages 1306–1314, 2007.

[15] T. Le-Anh and M. D. Koster. A review of design and control of automated guided vehicle systems. Research Paper 2004-030-LIS, RSM Erasmus University, 2004.

[16] R. Lochana Moorthy, W. Hock-Guan, N. Wing-Cheong, and T. Chung-Piaw. Cyclic deadlock prediction and avoidance for zone-controlled agv system. *International Journal of Production Economics*, 83(3):309–324, March 2003.

[17] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[18] openTCS. `http://www.opentcs.org`.

[19] S. A. Reveliotis. Conflict resolution in agv systems. *IIE Transactions*, 32:647–659, 2000.

[20] W. Schmid. *Kürzeste Wege in Strassennetzen mit Wegeverboten*. Verlag der Bayerischen Akademie der Wissenschaften, München, 2001.

[21] M. S. Waterman and T. F. Smith. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.