Swap Based Process Schedule Optimization using Discrete-Event Simulation

Maximilian Bügler, Gergő Dori & André Borrmann
Chair of Computational Modeling and Simulation – Technische Universität München

ABSTRACT: *Large construction projects usually involve many tasks, which are connected through dependencies and usage of common resources and materials. Determining the optimal order of task execution is in most of the cases impossible to do by hand. Therefore different methods for automatic optimization of large process schedules using a discrete-event simulation system were investigated. This paper introduces a new heuristic method for the resource constrained project scheduling problem, called swap-based optimization. Compared to creating an optimal schedule from scratch, the swap approach facilitates obtaining metrics about the performance of the result, before having worked through the entire construction process. Swaps are introduced into the simulation model by assigning priorities to the tasks. After running a simulation a list of possible swaps is created. Applying one of them and restarting the simulation will introduce a change into the sequence of the tasks within the schedule, generating a different schedule than the one before. Different tree search algorithms, traversing the space of possible swaps throughout a construction process, were analyzed. The suitability of the method is proven by an extensive case study.*

KEYWORDS: Resource Constrained Project Scheduling Problem, Project Schedule Optimization, Discrete-event Simulation, Task Swaps, Construction Site, Tree search.

1. Introduction

Creating schedules for large construction projects involves large numbers of tasks, their respective dependencies, resource constraints and material needs. Also geometric constraints can be taken into account (Marx et al., 2010). Deterministically solving such problems is considered NP-hard (Mingozzi et al., 1998). This paper introduces a new heuristic approach for solving the resource constrained project scheduling problem. In order to efficiently obtain feasible solutions for such a problem, discrete-event simulation can be used. While running a simulation it is possible to obtain a list of tasks that could be swapped while resulting in a new schedule. The proposed approach uses this possibility to create search trees based on those swaps and different search algorithms are investigated and compared to each other.

2. Discrete-Event Simulation

A constraint based discrete-event simulation model is used to generate schedules, since it always results in a feasible schedule (Beißert et al., 2007). A scenario is defined by a set of tasks, respective prerequisites, and resource and material requirements. First a list of tasks with no prerequisites is generated. Then one or more of

those tasks are executed, depending on the availability of the required resources and materials. The tasks may be prioritized for the selection process. After that the time of the next event is determined by calculating the time the next task is finished. Then a new list of tasks, which have satisfied precedence and resource constraints, is created. This procedure is repeated until all tasks have been executed. The result is a feasible schedule providing metrics that can be used for optimization.

3. Swap Concept

After creating an initial schedule using the discrete-event simulator, where tasks are randomly selected from the list of executable tasks, a list of possible swaps is created. Each time a task is executed during simulation, it was selected from a list of executable tasks. At this point the list is updated and some other tasks may have to be removed from the list because required resources are now used by the executed task. When this occurs it is possible to swap the executed task with each task that was removed. Over the entire span of the simulation this will result in a list of possible swaps for each time step. In order to apply swaps in the simulation the tasks are prioritized. For the base schedule all priorities are selected randomly, or by incrementally numbering all processes. When a swap of two tasks is applied the respective priorities are flipped. In the following simulations the tasks are no longer selected randomly but based on their respective priorities. Figure 1 shows an illustrative case for a simple project, where arrows indicate dependencies and colors indicate resource usage. Hence tasks B and C use the same resource, so cannot be executed simultaneously. Also, they must be executed after task A has finished. Similarly task D needs task C to be finished. An obvious solution for an execution order for this problem is illustrated in Figure 2. When applying a swap of tasks B and C, it can be observed that the schedule is shorter, as it is presented in Figure 3. This shows how the application of swaps can influence the overall project schedule duration.

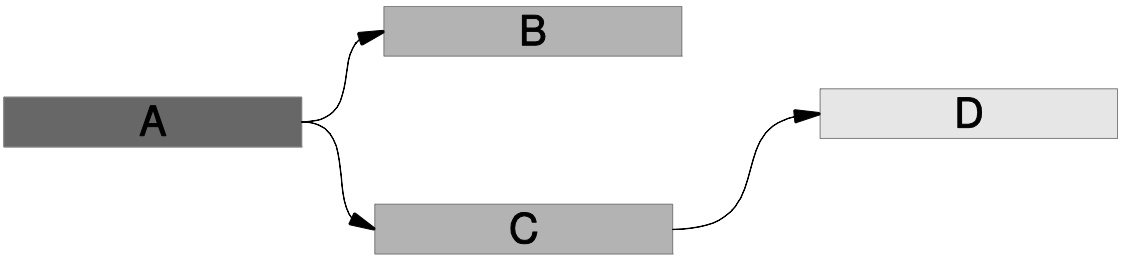

Figure 1: Dependencies of illustrative case. Colors indicate resource usages and arrows indicate predecessor constraints.

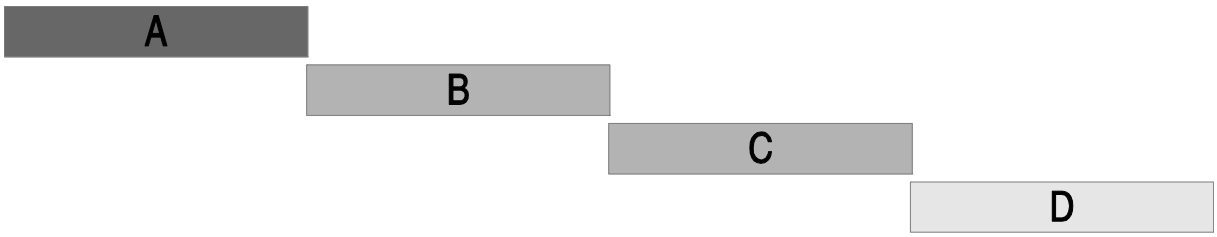

Figure 2: Possible execution schedule

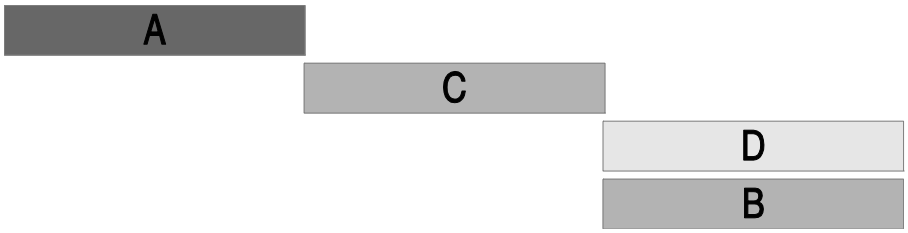

Figure 3: Execution schedule after swapping B and C

A key advantage of using the swap based approach, compared to building a schedule from scratch, is that simulating the resulting prioritization provides a schedule duration which can be used as a precise score for tree search algorithms.

4. Search Approaches

Given a list of possible swaps a search tree can be constructed which covers all possible schedules (see Figure 4). In practical cases with several thousand tasks though, this tree will grow very large and may therefore be infeasible to search completely. Especially when considering that each application of a swap requires a simulation run to determine the resulting schedule duration. Therefore different ways to traverse the resulting trees have been investigated, using a variety of pruning methods. Pruning describes the removal of specific branches of search trees, according to some criteria (Mansour, 1997). Since the optimization starts from a single random solution a single tree branching out from this initial solution (root node) is created. Each child of the root node represents the application of a single swap. Each further level down the tree incorporates additional swaps. The simplest possible approach is greedy search is to only look for the most beneficial swap at each level, if such a swap exists (Black et al., 2005). Hence the branching only takes place on the single most beneficial swap, after which this procedure is repeated until no more improvements can be found. The downside of this approach is that there are cases where a non-beneficial swap allows for another swap that will result in a shorter duration as the initial solution. Such a case is illustrated in Figure 5, which shows dependencies between five tasks A through E. A schedule created from these dependencies is illustrated in Figure 6. Applying a swap between tasks B and C increases the execution duration significantly by preventing any parallel execution of tasks. This is shown in Figure 7. The additional application of a swap between A and E now allows parallel execution of A and C and therefore results in a shorter execution time compared to the initial schedule of Figure 6. The final schedule is illustrated in Figure 8.

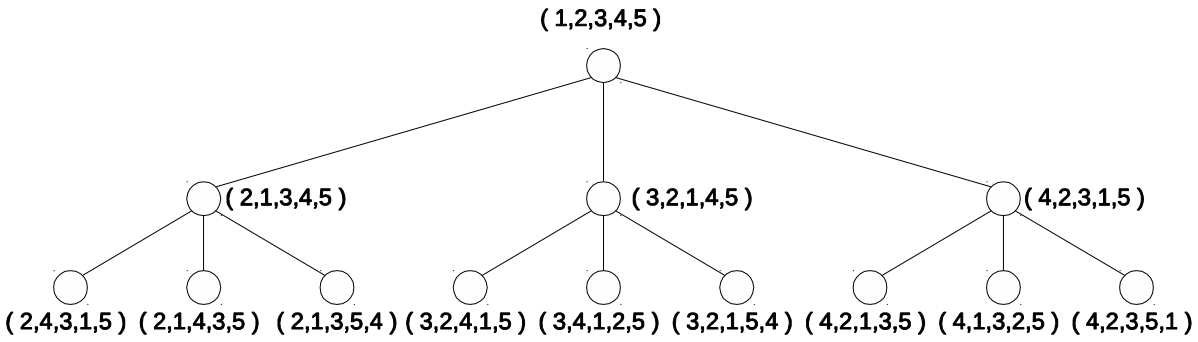

Figure 4: Search Tree Illustration. The vectors contain the priorities of the tasks.

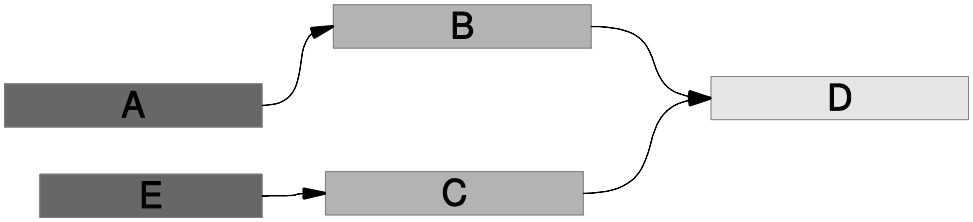

Figure 5: Dependencies of illustrative case 2. Colors indicate resource usages and arrows indicate predecessor constraints.

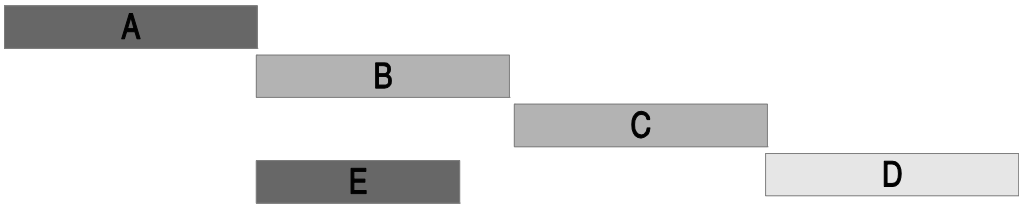

Figure 6: Possible execution schedule

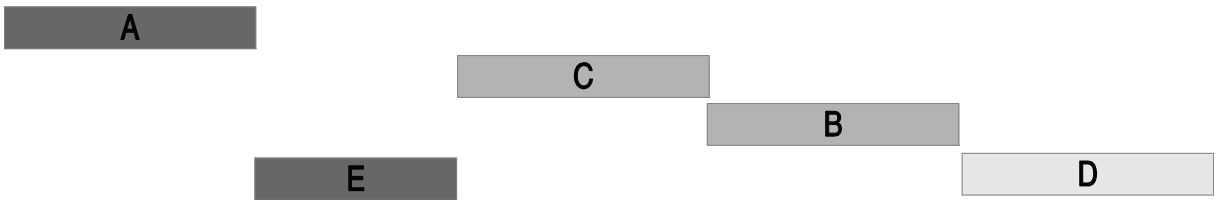

Figure 7: Longer schedule after applying one swap of B and C

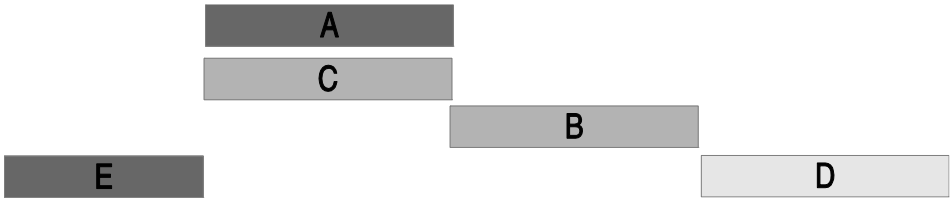

Figure 8: Shorter schedule after applying swaps between B, C and A, E

In order to incorporate solutions where multiple swaps, which each at its own are non-beneficial, applied together give a better solution a less radical way to prune the search tree is required. A simple measure to realize this is by introducing a tolerance factor. This means that instead of stopping when no more beneficial swaps can be found, also swaps which increase the overall execution by a certain tolerance amount are taken into account. While in the case illustrated in Figure 5 this tolerance amount would need to be quite large, the ratio by which the swap prolongs the schedule in larger problems is expected to be much smaller. A problem arising with this approach is that there are many swaps that may result in delays below the tolerance ratio, and the search tree therefore may expand quite strongly. Therefore it will in large cases become infeasible to perform a full search of the tree. A useful assumption would be that longer series of non-beneficial swaps become less likely to yield a benefit. This assumption allows splitting the search tree into segments of limited depth. For instance assuming that there is no beneficial series of more than k non-beneficial swaps would allow only searching the tree to a depth of k while being very tolerant. Then the best solution from the limited-depth search tree is selected and a new search tree is built, using the solution as a new root node. Each segment of this search tree will therefore be limited to n^k leaf nodes, where n is the number of possible swaps.

Another way to traverse the search tree is simulated annealing, which is inspired by the solidification of matter when cooling down (König et al., 2009). This approach was modeled by introducing a temperature value starting at a value of 1 and reducing over time by a certain decay rate, between 0 and 1. The decay is applied after each depth level in the implementation used. The temperature value is used as the probability of accepting an inferior swap. It can therefore be interpreted as tolerance. At some point no more change takes place, when the process can be stopped.

An issue that generally can arise during traversal of the resulting search trees is revisiting the same state multiple times. It is of course desirable to prevent this from happening. Therefore tabu-search was applied to the search processes. Tabu-search means verifying whether a new state is practically equal to a previously encountered state and therefore investigating it is redundant (Glover et al., 1989).

5. Case study

In order to test and compare the proposed methods a case study has been performed. The scenario that was analysed consists of two connected walls of drill pines and and four excavation areas. Each drill pine needs three tasks to be built. First a hole needs to be drilled. This task requires one worker to operate the drill. Afterwards three

workers and a crane are necessary to insert the reinforcment cage into the hole. Furthermore two workers and a concrete pump are required to fill the hole with concrete. Finally the top of the pine needs to be grinded flat, requiring one worker and a grinding machine. Since the pines are overlapping, the cropped pines, need to be placed before the others. When all drillpines connected to an excavation area are finished, the excavation can be started. This requires one worker and an excavator to be available. The scenario is finished when all tasks are executed. Additionally the areas blocked by the tasks and the employed machines are taken into account during simulation. The scenario is illustrated in Figure 9. In order to evaluate the results of the tested approaches, a Mixed-Integer-Linear-Programming (MILP) formulation of the problem has been solved to obtain an exact solution to the problem (Konéa et al., 2011). The optimal solution is illustrated in Figure 12, located in the appendix.

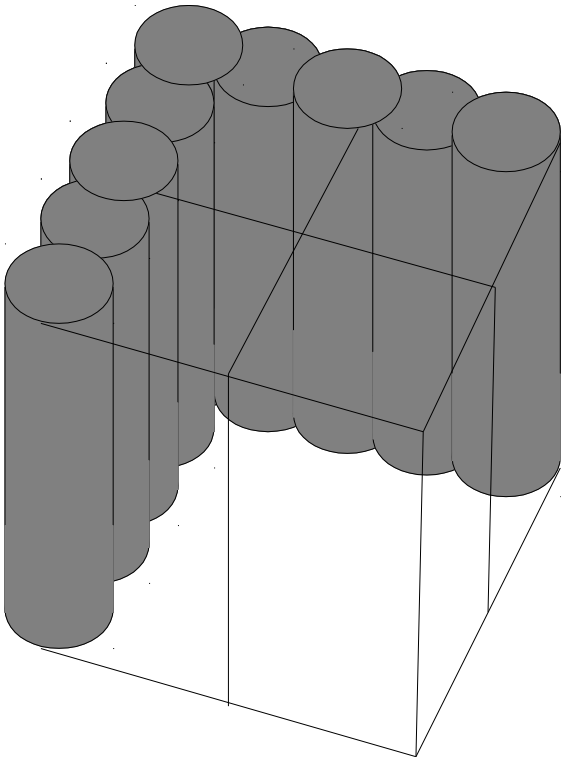

Figure 9: Scenario used for Case Study. The grey cylinders represent drill pines and the squares represent the four excavation areas.

As an objective, the minimization of the overall execution time of the resulting schedule was used. It would also be possible to use any other metric obtained through simulation for the search algorithms. Table 1 shows the results obtained from the different algorithms. The processing times were all measured using an Intel Core i7-3520M CPU while utilizing all cores. The greedy approach is very fast, but gives suboptimal results. Next simulated annealing was tested using different levels of decay, where 1 is no decay and 0 is immediate decay, so equal to the greedy method. The optimal solution was found with decay values of 0.7 and higher, requiring 21 minutes of processing time. But also with strong decay, the result was near optimal and the processing time was very low. The

limited depth series approach returned the optimal solution starting at depth 3, so it was not necessary to apply more than 3 non beneficial swaps in this scenario, to find the optimal solution. Finally the tree search with tolerance was evaluated with different tolerance levels of up to 10%. The optimal solution was not found here. The comparisons of the performances are illustrated in Figure 10 and Figure 11.

The limited depth series approach required the lowest number of simulations and therefore was fastest in finding the optimal solution for the problem, followed by the simulated annealing algorithm. The tolerant search approach was increasing in complexity very quickly. With tolerance values above 10% the processing time exceeded several hours. But the result returned with 10% tolerance is not very far off the optimal solution and was processed very quickly, so this might still be a reasonable heuristic.

Table 1: Comparison of search algorithms

Algorithm	Parameter	Resulting Schedule duration	Number of Simulations performed	Required time
Greedy		147h	117	746ms
Simulated Annealing	Decay=0.4	119h	11903	2.8s
Simulated Annealing	Decay=0.6	119h	674561	73.7s
Simulated Annealing	Decay=0.7	118h	11264258	1255.7s
Limited Depth Series	Depth=1	147h	39	476ms
Limited Depth Series	Depth=2	123h	3189	2s
Limited Depth Series	Depth=3	118h	85171	11.8s
Limited Depth Series	Depth=4	118h	3616705	384.3s
Tolerant search	Tolerance=1%	147h	197	712ms
Tolerant search	Tolerance=5%	123h	6064	4.9s
Tolerant search	Tolerance=10%	123h	192245	21.8s
MILP Solver		118h		10h

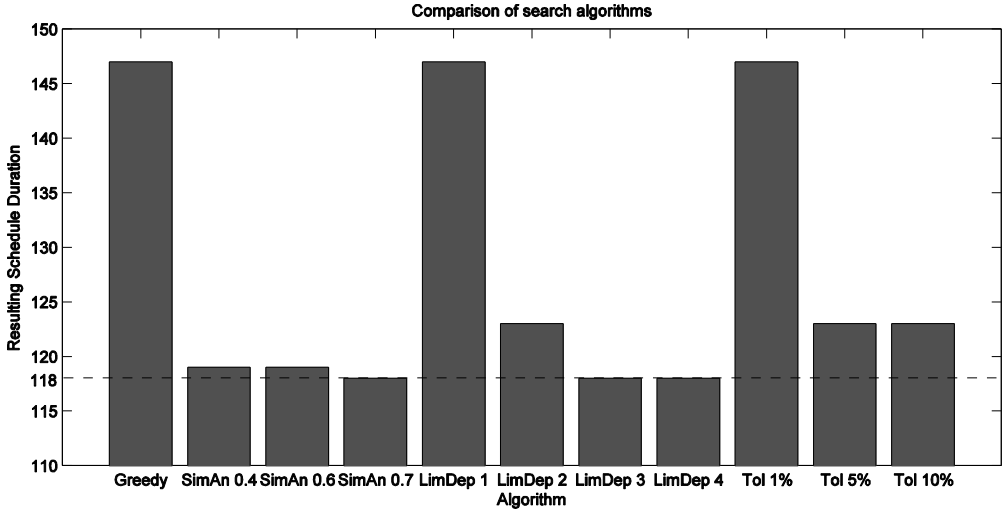

Figure 10: Result comparison of search algorithms

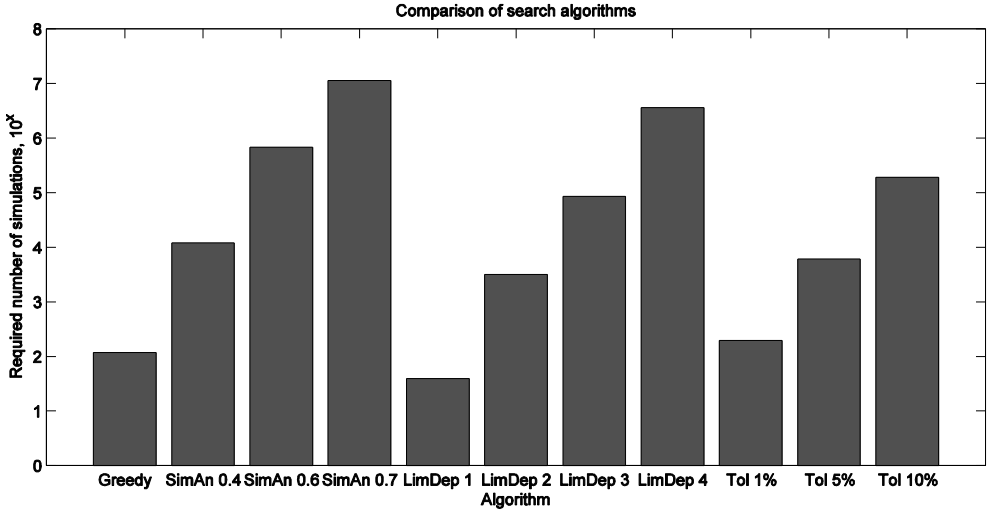

Figure 11: Number of simulations needed for the different search algorithms

6. Pitfalls

A major pitfall encountered during analysis of the approach was the necessity of using tabu-search. Due to the huge number of simulations that need to be performed, the memory demands for book keeping of the evaluated states was very high. Furthermore the application of a swap can drastically change the entire schedule after the time of the swap. This can cause the algorithms to randomly traverse the search space, making pruning more difficult. In the scenarios that were evaluated so far, this did not impose a major obstacle, though.

7. Conclusions

This paper introduced a new heuristic approach for solving the resource constraint project scheduling problem. Search trees were constructed from possible task swaps, obtained through constraint based discrete-event simulation. The proposed approach returned very good results in the studied case. Among the algorithms that were tested, simulated annealing and the limited depth series search performed best and returned the optimal solution in much shorter time than the MILP approach used for benchmarking. Though there can be cases where the tolerant search algorithm would also return good results. In order to evaluate this, further research is required. More complex case studies will be performed to test whether the results are consistent among different kinds of projects. Further experiments are required to determine the number of non-beneficial swaps that can still be beneficial in large scale projects.

8. References

Beißert U., König M., Bargstädt H.-J. (2007). Constraint-Based Simulation of Outfitting Processes in Building Engineering, in: *Proceedings of CIB w78 Conference, Maribor, Slovenia*

Black, P. E. (2005). Greedy Algorithm. Published electronically, *Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology (NIST).*

Dori G., Borrmann A. (2011). Automatic generation of Complex Bridge Construction Animation Sections by coupling Constraint-based Discrete-Event Simulation with Game Engines. *Proceedings of CONVR - International Conference on Construction Applications of Virtual Reality, Weimar, Germany, 2011*

Glover F. (1989). Tabu Search - Part 1. *ORSA Journal on Computing 1 (2): 190–206.*

Konéa O., Artiguesa C., Lopeza P., Mongeauc M. (2011). Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research Volume 38, Issue 1, January 2011, Pages 3–13*

König M., Beißert U. (2009). Construction Scheduling Optimization by Simulated Annealing. *Proceedings of the 26th International Symposium on Automation and Robotics in Construction (ISARC 2009)*

Mansour, Y. (1997). Pessimistic decision tree pruning based on tree size. *Machine Learning – International Workshop then Conference. Pages 195-201*

Marx A., Erlemann K., König M. (2010). Simulation of Construction Processes considering Spatial Constraints of Crane Operations. *Proceedings of the XIIIth International Conference on Computing in Civil and Building Engineering*

Mingozzi A., Maniezzo V., Ricciardelli S. Bianco L. (1998). An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science May 1998 vol. 44 no. 5 714-729*

Robinson S. (2004). Simulation - The practice of model development and use. *Wiley, p. 15-24*

9. Appendix

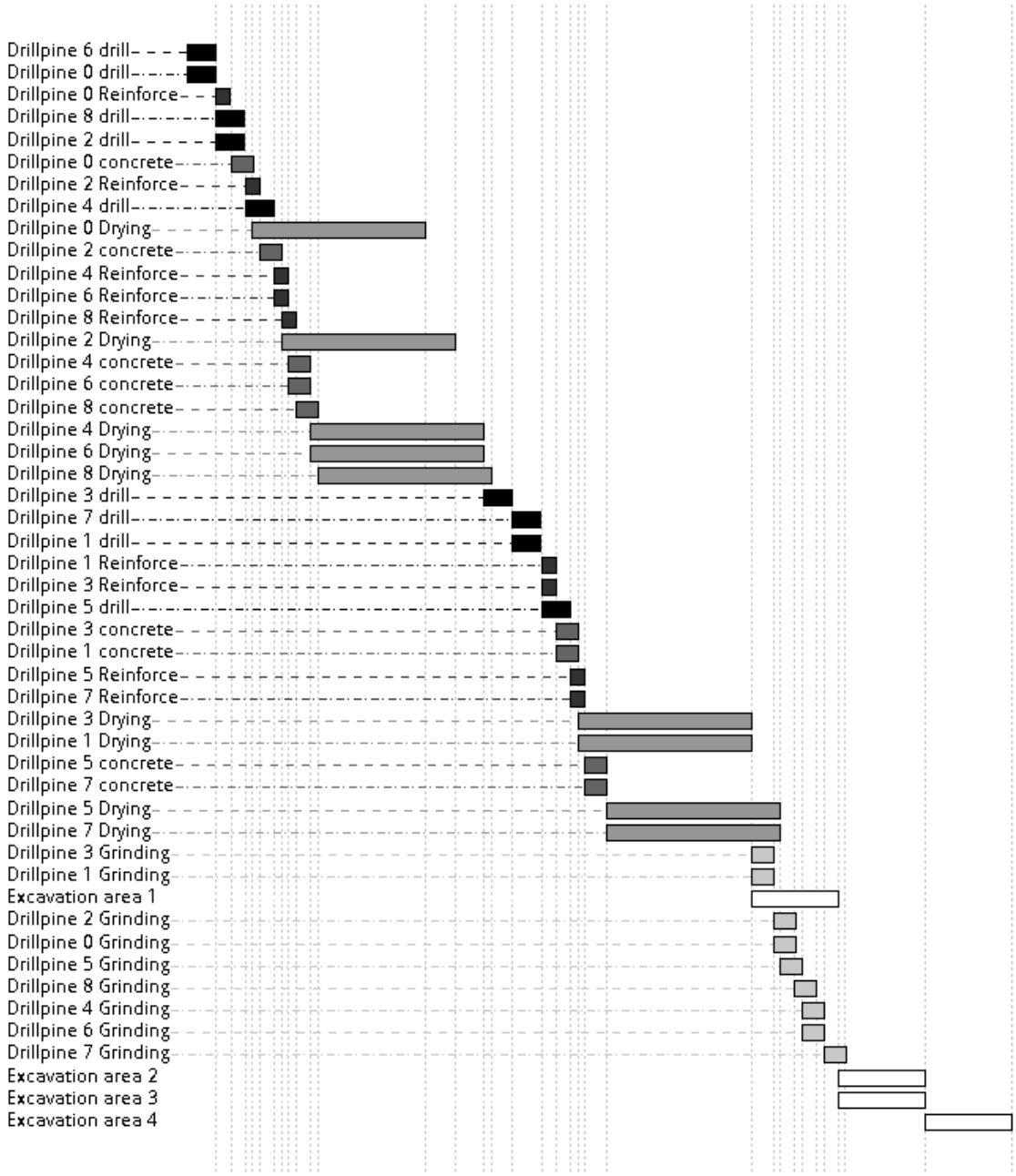

Figure 12: Optimal Schedule for Case Study Scenario