applied sciences

MDPI

Article

Job Shop Scheduling Problem Optimization by Means of Graph-Based Algorithm

Jiri Stastny [1,2,*], Vladislav Skorpil [3], Zoltan Balogh [4] and Richard Klein [1]

[1] Institute of Automation and Computer Science, Brno University of Technology, 616 69 Brno, Czech Republic; 185649@vutbr.cz
[2] Department of Informatics, Mendel University in Brno, 613 00 Brno, Czech Republic
[3] Department of Telecommunications, Brno University of Technology, 616 00 Brno, Czech Republic; skorpil@feec.vutbr.cz
[4] Computer Science Department, Constantine the Philosopher University in Nitra, 949 74 Nitra, Slovakia; zbalogh@ukf.sk
* Correspondence: stastny@fme.vutbr.cz

Abstract: In this paper we introduce the draft of a new graph-based algorithm f or optimization of scheduling problems. Our algorithm is based on the Generalized Lifelong Planning A* algorithm, which is usually used for path planning for mobile robots. It was tested on the Job Shop Scheduling Problem against a genetic algorithm's classic implementation. The acquired results of these experiments were compared by each algorithm's required time (to find the best solution) as well as makespan. The comparison of these results showed that the proposed algorithm exhibited a promising convergence rate toward an optimal solution. Job shop scheduling (or the job shop problem) is an optimization problem in informatics and operations research in which jobs are assigned to resources at particular times. The makespan is the total length of the schedule (when all jobs have finished processing). In most of the tested cases, our proposed algorithm managed to find a solution faster than the genetic algorithm; in five cases, the graph-based algorithm found a solution at the same time as the genetic algorithm. Our results also showed that the manner of priority calculation had a non-negligible impact on solutions, and that an appropriately chosen priority calculation could improve them.

Keywords: Genetic algorithms; graph-based algorithm; Job Shop Scheduling Problem; optimization

Citation: Stastny, J.; Skorpil, V.; Balogh, Z.; Klein, R. Job Shop Scheduling Problem Optimization by Means of Graph-Based Algorithm. *Appl. Sci.* **2021**, *11*, 1921. https://doi.org/10.3390/app11041921

Academic Editor: Alexandre Carvalho

Received: 11 January 2021
Accepted: 17 February 2021
Published: 22 February 2021

1. Introduction

Nowadays, graph-based algorithms are a popular tool for solving various optimization problems. However, most optimization methods are only used for solving one type of task. On the other hand, methods for optimization of several problems, in most cases, lack sufficient computing power. There are a wide variety of different algorithms available for the optimization of scheduling problems. One of the most well-known is the genetic algorithm (GA).

In this article, we introduce the draft of a graph-based algorithm for scheduling optimization problems. The proposed algorithm aims to have the flexibility and scope of application of advanced optimization methods—such as genetic algorithms—and to retain most of the positive characteristics of advanced graph algorithms.

We researched the literature carefully and paid great attention—though the references contained mainly general literature reviews. Our proposed algorithm appears unique; we could not find any other reference with comparable results for the given problem.

There are many different methods, algorithms and modifications, some of which already contain some of the required properties. Their structure allows for further adaptation. Additionally, some algorithms are directly intended for further development into new algorithms. From various graph-based algorithms, the Forward Version of Generalized

Lifelong Planning A* (GLPA*) [1] —used for path planning in mobile robot navigation— was chosen. We then attempted to modify it for use in other types of optimization and tested it on various travelling salesman problems against the genetic algorithm (GA) [2,3].

The solution was based on a new algorithm, a hybrid between graph-based and genetic algorithms. An algorithm was designed and implemented that combines the advantages of graph-based and genetic algorithms. We chose the genetic algorithm for comparison because it was suited for this purpose and remains one of the most well-known algorithms for optimization in the planning area. The key parameters for the solution in the given area were makespan and calculation speed. Optimization according to other parameters would be possible; however, given the importance of the above two key parameters, it would not make sense.

The task of developing new optimization procedures resulted from a general effort to optimize problem solving. Our proposed graph-based algorithm has been successfully tested on other problems (see references) and was more successful that the commonly-used genetic algorithm in solving the Job Scheduling. The motivation for the development of new algorithms was to obtain faster, more accurate results in the given area.

In this paper, the term "performance" refers to the success of our algorithm according to the following parameters: makespan and calculation speed. Using those parameters, our new graph-based algorithm was compared with a genetic algorithm (for makespan differences, see conclusion; for calculation speed, see Table 1). These parameters were chosen because they are key, and because, according to references, measuring other parameters would have made little sense. For the industry, improvements in makespan and calculation speed are economically-driven (e.g., seeking a reduction in price).

The graph-based algorithm had a better average deviation from best-known makespans than the GA. The average deviations of the best-known makespans from the best makespans were as follows: 5.19% for GA and 4.7% for α; 5.05% for β and 5.08% for γ, for the developed (graph-based) algorithm. The average calculation speed of the proposed graph algorithm is two to three times higher than the GA.

When researching the new graph-based algorithm, we proceeded through five steps: (1) analysis of graph-based algorithms, (2) analysis of genetic algorithms, (3) development and design of new algorithm, (4) verification of algorithm on job shop scheduling problem (JSSP), and (5) comparison with the genetic algorithm used in the given area. Let us take a look at our new algorithm's performance on job shop scheduling problems.

Table 1. Comparison of algorithms on job shop scheduling problem (JSSP) by time and makespan.

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
10	5	la01	666	90	Graph-based α	666	00:00.04	678.18	00:07.38
					Graph-based β	666	00:00.04	683.79	00:06.54
					Graph-based γ	666	00:00.03	682.54	00:03.83
					Genetic	666	00:00.99	666.19	00:28.65
10	5	la02	655	90	Graph-based α	672	00:00.03	728.8	00:17.91
					Graph-based β	709	00:00.02	773.09	00:06.95
					Graph-based γ	678	00:00.02	769.4	00:06.29
					Genetic	655	00:04.44	686.67	00:50.59
10	5	la03	597	90	Graph-based α	617	00:00.04	655.23	00:11.06
					Graph-based β	609	00:00.03	657.66	00:06.73
					Graph-based γ	617	00:00.03	666.61	00:03.55
					Genetic	613	00:02.41	638.2	00:50.81
10	5	la04	590	90	Graph-based α	604	00:00.04	634.17	00:09.94
					Graph-based β	595	00:00.04	641.77	00:09.34
					Graph-based γ	600	00:00.02	643.07	00:06.08
					Genetic	595	00:01.32	612.89	00:48.42
10	5	la05	593	90	Graph-based α	593	00:00.02	593	00:00.04
					Graph-based β	593	00:00.02	593.27	00:00.74
					Graph-based γ	593	00:00.02	593.24	00:00.11
					Genetic	593	00:00.01	593	00:00.12

Table 1. *Cont.*

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
15	5	la06	926	120	Graph-based α	926	00:00.06	926.1	00:01.22
					Graph-based β	926	00:00.06	926.59	00:02.22
					Graph-based γ	926	00:00.06	926.27	00:02.18
					Genetic	926	00:00.04	926	00:03.41
15	5	la07	890	120	Graph-based α	890	00:00.12	915.83	00:34.05
					Graph-based β	903	00:00.07	957.15	00:13.01
					Graph-based γ	903	00:00.08	957.43	00:08.24
					Genetic	890	00:09.89	909.04	01:10.46
15	5	la08	863	120	Graph-based α	863	00:00.08	869.58	00:18.09
					Graph-based β	863	00:00.07	886.27	00:09.32
					Graph-based γ	863	00:00.07	883.58	00:11.99
					Genetic	863	00:00.12	864.28	00:45.89
15	5	la09	951	120	Graph-based α	951	00:00.07	951	00:00.10
					Graph-based β	951	00:00.08	951.48	00:00.45
					Graph-based γ	951	00:00.06	951.28	00:01.40
					Genetic	951	00:00.03	951	00:00.81
15	5	la10	958	120	Graph-based α	958	00:00.07	958	00:00.10
					Graph-based β	958	00:00.10	958	00:00.15
					Graph-based γ	958	00:00.06	958	00:01.33
					Genetic	958	00:00.03	958	00:00.27
20	5	la11	1222	150	Graph-based α	1222	00:00.14	1222	00:00.82
					Graph-based β	1222	00:00.14	1224.72	00:07.29
					Graph-based γ	1222	00:00.13	1224.72	00:10.09
					Genetic	1222	00:00.06	1222	00:08.97

Table 1. *Cont.*

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
20	5	la12	1039	150	Graph-based α	1039	00:00.14	1039.3	00:00.75
					Graph-based β	1039	00:00.13	1040.11	00:03.03
					Graph-based γ	1039	00:00.13	1039.13	00:00.51
					Genetic	1039	00:00.05	1039	00:09.49
20	5	la13	1150	150	Graph-based α	1150	00:00.13	1150	00:00.18
					Graph-based β	1150	00:00.14	1150.25	00:02.55
					Graph-based γ	1150	00:00.13	1150	00:01.27
					Genetic	1150	00:00.04	1150	00:03.73
20	5	la14	1292	150	Graph-based α	1292	00:00.14	1292	00:00.16
					Graph-based β	1292	00:00.13	1292	00:00.17
					Graph-based γ	1292	00:00.13	1292	00:00.18
					Genetic	1292	00:00.04	1292	00:00.06
20	5	la15	1207	150	Graph-based α	1221	00:00.18	1284.92	00:53.47
					Graph-based β	1235	00:00.13	1341.45	00:38.72
					Graph-based γ	1246	00:00.14	1344.66	00:42.69
					Genetic	1227	00:01.69	1270.59	01:33.41
10	10	la16	945	120	Graph-based α	982	00:00.14	1057.65	00:31.06
					Graph-based β	994	00:00.13	1068.95	00:17.09
					Graph-based γ	982	00:00.19	1071.77	00:19.66
					Genetic	979	00:01.57	1022.91	01:11.39
10	10	la17	784	120	Graph-based α	793	00:00.17	835.65	00:23.33
					Graph-based β	793	00:00.17	841.72	00:14.33
					Graph-based γ	793	00:00.14	842.9	00:17.10
					Genetic	795	00:00.13	832.96	00:55.36

Table 1. *Cont.*

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
10	10	la18	848	120	Graph-based α	849	00:00.14	917.28	00:22.30
					Graph-based β	861	00:00.19	920.34	00:16.73
					Graph-based γ	861	00:00.13	921.51	00:20.68
					Genetic	871	00:00.16	914.83	00:47.57
10	10	la19	842	120	Graph-based α	877	00:00.13	934.03	00:21.21
					Graph-based β	889	00:00.13	951.43	00:15.39
					Graph-based γ	886	00:00.11	954.89	00:13.99
					Genetic	875	00:02.52	925.92	01:02.01
10	10	la20	902	120	Graph-based α	921	00:00.15	976.8	00:22.44
					Graph-based β	914	00:00.14	967.72	00:18.57
					Graph-based γ	907	00:00.12	960.93	00:23.14
					Genetic	924	00:00.68	976.04	00:57.70
15	10	la21	1046	150	Graph-based α	1144	00:01.46	1201.86	00:54.14
					Graph-based β	1122	00:00.51	1230.36	00:42.71
					Graph-based γ	1149	00:00.48	1230.86	00:46.21
					Genetic	1140	00:06.32	1218.33	01:15.78
15	10	la22	927	150	Graph-based α	999	00:00.50	1093.93	00:44.53
					Graph-based β	1051	00:00.53	1109.53	00:44.76
					Graph-based γ	1027	00:01.69	1113.67	00:38.11
					Genetic	1029	00:08.06	1105.7	01:25.18
15	10	la23	1032	150	Graph-based α	1068	00:00.43	1131.96	00:37.00
					Graph-based β	1076	00:00.35	1144.07	00:31.82
					Graph-based γ	1086	00:00.36	1141.74	00:33.06
					Genetic	1046	00:01.64	1140.26	01:12.26

Table 1. *Cont.*

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
15	10	la24	935	150	Graph-based α	1013	00:00.41	1070.53	00:55.55
					Graph-based β	1029	00:01.64	1075.82	00:40.20
					Graph-based γ	992	00:00.54	1082.89	00:37.05
					Genetic	1024	00:01.16	1088.24	01:07.66
15	10	la25	977	150	Graph-based α	1079	00:00.50	1125.75	00:38.39
					Graph-based β	1076	00:00.50	1150.38	00:44.80
					Graph-based γ	1083	00:00.54	1155.49	00:34.43
					Genetic	1060	00:03.24	1155.14	01:17.27
20	10	la26	1218	180	Graph-based α	1311	00:00.84	1395.86	01:06.53
					Graph-based β	1340	00:02.17	1405.4	01:27.17
					Graph-based γ	1328	00:04.80	1411.57	01:11.35
					Genetic	1340	00:04.13	1428.12	01:36.13
20	10	la27	1235	180	Graph-based α	1368	00:00.79	1459.38	01:00.30
					Graph-based β	1390	00:01.04	1486.95	01:24.53
					Graph-based γ	1379	00:01.06	1483.46	01:04.60
					Genetic	1411	00:04.29	1485.78	01:38.00
20	10	la28	1216	180	Graph-based α	1328	00:00.87	1399.72	01:04.58
					Graph-based β	1349	00:00.71	1437.18	00:53.57
					Graph-based γ	1348	00:00.90	1435.36	00:57.33
					Genetic	1364	00:01.27	1431.87	01:28.75
20	10	la29	1157	180	Graph-based α	1320	00:00.77	1401.5	00:58.80
					Graph-based β	1316	00:01.97	1402.03	01:13.52
					Graph-based γ	1326	00:00.76	1404.5	01:15.13
					Genetic	1350	00:05.84	1435.2	01:26.83

Table 1. *Cont.*

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
20	10	la30	1355	180	Graph-based α	1467	00:00.83	1549.21	00:43.81
					Graph-based β	1458	00:00.90	1560.17	00:47.64
					Graph-based γ	1481	00:00.80	1558.42	00:55.33
					Genetic	1472	00:04.83	1544.5	01:47.72
20	10	la31	1784	180	Graph-based α	1784	00:06.40	1837.23	02:10.06
					Graph-based β	1784	00:08.73	1841.66	02:39.27
					Graph-based γ	1784	00:14.42	1835.87	02:47.32
					Genetic	1807	00:16.38	1898.83	02:18.27
30	10	la32	1850	240	Graph-based α	1851	00:13.64	1914.26	02:41.29
					Graph-based β	1850	00:10.02	1917.62	02:39.39
					Graph-based γ	1850	00:10.87	1921.69	02:30.43
					Genetic	1895	00:16.50	1993.32	02:05.09
30	10	la33	1719	240	Graph-based α	1739	00:02.39	1796.38	02:08.04
					Graph-based β	1725	00:05.58	1817.67	02:44.53
					Graph-based γ	1744	00:02.56	1819.53	02:36.41
					Genetic	1780	00:08.46	1845.43	02:12.57
30	10	la34	1721	240	Graph-based α	1819	00:02.29	1878.34	01:51.83
					Graph-based β	1797	00:02.55	1889.43	02:11.28
					Graph-based γ	1819	00:02.33	1897.38	01:57.48
					Genetic	1846	00:11.64	1909.86	02:04.97
30	10	la35	1888	240	Graph-based α	1980	00:02.43	2115.15	02:32.74
					Graph-based β	2067	00:09.87	2197.11	02:20.97
					Graph-based γ	2078	00:06.73	2193.47	02:36.02
					Genetic	2021	00:17.32	2143.59	02:48.95

Table 1. *Cont.*

Jobs	Machines	Instance	Best Known Makespan	Time of Optimization	Algorithm Used	Makespan	Time of Finding Best Makespan [mm:ss]	Average Makespan	Average Time of Finding Makespan [mm:ss]
15	15	la36	1268	180	Graph-based α	1427	00:00.99	1494.55	00:51.76
					Graph-based β	1387	00:00.99	1511.63	00:48.84
					Graph-based γ	1419	00:01.07	1513.32	01:01.04
					Genetic	1373	00:02.11	1482.9	01:30.90
15	15	la37	1397	180	Graph-based α	1576	00:01.03	1667.02	01:16.16
					Graph-based β	1546	00:05.63	1664.63	01:20.99
					Graph-based γ	1565	00:04.33	1663.85	01:24.34
					Genetic	1549	00:08.89	1681.12	01:48.09
15	15	la38	1196	180	Graph-based α	1375	00:00.99	1448.17	00:52.06
					Graph-based β	1381	00:00.99	1446.89	01:02.95
					Graph-based γ	1378	00:01.15	1449.01	01:04.43
					Genetic	1397	00:01.46	1469.16	01:19.78
15	15	la39	1233	180	Graph-based α	1385	00:01.04	1455.54	01:00.09
					Graph-based β	1377	00:00.99	1452.02	01:21.56
					Graph-based γ	1371	00:01.57	1454.57	01:12.73
					Genetic	1405	00:01.16	1475.14	01:20.90
15	15	la40	1222	180	Graph-based α	1386	00:01.50	1446.41	01:16.11
					Graph-based β	1356	00:03.92	1435.03	01:25.09
					Graph-based γ	1364	00:05.21	1438.51	01:34.75
					Genetic	1412	00:10.56	1484.72	01:40.72

2. Job Shop Scheduling Problem

The job shop scheduling problem (JSSP) is about allocating limited resources to a variety of tasks for the purpose of obtaining optimal scheduling solutions. These optimized scheduling solutions are important for achieving an efficient and orderly production process [4]. The JSSP is one of the most well-known problems in the fields of production management and combinatorial optimization. The classical n-by-m JSSP can be described as the flow shop problem generalization or the open shop problem specification: scheduling n jobs on m machines with the objective of minimizing the scheduled completion time (makespan) of all jobs. Each of n jobs consists of several operations on m specified machines, which need an uninterrupted processing time of a given length. Operations of the same job can be performed only on one machine at a time and each job must be processed on each machine exactly once [5].

In order to increase production efficiency, reduce cost and/or improve product quality, efficient methods for solving the JSSP are required. Additionally, the JSSP is acknowledged as one of the most challenging NP-hard problems—and no algorithms that can solve the JSSP consistently, even on small scale problems, are known to exist [5].

Nowadays, with the usage of metaheuristic algorithms, new ways of obtaining better results than were possible with classical methods (such as greedy or dispatching heuristic algorithms) have emerged. These metaheuristic algorithms are widely used because of their flexibility and global optimization capabilities [4]. Algorithms for the solution of the JSSP include population-based metaheuristic ones, including particle swarm optimization (detailed description in [6,7]), genetic algorithms, artificial immune systems and their hybrids (see [8–10]), construction and improvement heuristics including taboo search (described in [11–13]) and simulated annealing (described in [14,15]). Also, heuristic methods can be used for large-scale problems. These include dispatching priority rules (see [16–18]), the shifting bottleneck approach (detailed description in [19–21]) and Lagrangian relaxation (see [22–24]) [5].

However, each of these methods comes with their own drawback. Simulated annealing has the tendency to generate poor designs that have a high chance of missing local minima; its convergence is also quite slow. The taboo search uses the serial search method and is low-efficiency. Particle swarm optimization has the tendency to become trapped in local minima. Exact techniques, such as branch and bound (described in [25,26]) and dynamic programming (see [27,28]) can be used to solve only modest-scale problems, because of the complexity of the JSSP [5].

As the genetic algorithm can search efficiently through large search spaces and does not explicitly require additional information about the objective function in order to be optimized, it has been applied to many combinatorial problems, including scheduling [29]. Moreover, its architecture makes it well suited for parallel computation. On the other hand, the GA considers only one new solution at a time [4].

3. Algorithm Description

The proposed algorithm was developed based on the Generalized Lifelong Planning A*, which is the forward algorithm and framework that generalizes Lifelong Planning A* (LPA*) (described in detail in [30]) and its non-deterministic version of the Minimax LPA* (described in detail in [31]). Additionally, it is designed to allow further development of efficient versions of LPA* and Minimax LPA*, including versions that use inconsistent heuristics or good tiebreaking. That is described in detail in [32].

There are only two differences between LPA* and GLPA* algorithms—and those differences are the reason why GLPA* was chosen over the classical implementation of LPA*. One difference is that GLPA* generalizes the calculation of RHS values and priorities, which enables a considerable amount of adaptability in determination of the optimization criterion and the order in which new states are expanded. Another difference is that the priority queue of GLPA* does not contain all locally inconsistent states—only locally

inconsistent states that have not yet been expanded as overconsistent during the current run of the function ComputePlan(). This difference can considerably speed up a search [32].

The first major modification made was the change in data representation. This change was heavily inspired by the genetic algorithm. Although the data representation in GLPA* is very suitable for solving path planning problems, it is less effective for other types of optimization—and for some problems, it is completely useless. For this reason, data of the proposed algorithm is now represented by the usage of bit strings. In most cases, this means that numerical parameters are represented by integers. When solving complex problems, a representation using Gray code can be used, which causes even greater changes in data of possible solutions when expanding new states and further reduces the possibility of converging on local extremes. Real numbers usage is not recommended, because it tends to cause problems for the expansion of new states [3].

The second significant change we made stems from the first; since the algorithm uses a different data representation, the values of newly acquired states may no longer depend on the values of their predecessors. Furthermore, the calculation of the RHS value may now vary depending on the problem—meaning the old way of calculation is not always possible. Therefore, a new function—StateEvaluation() [2]—was introduced, which calculated the RHS value of the given problem. This is basically an objective function used to determine how close the obtained solution is in order to achieve the goals of optimization. In order to prevent the algorithm from converging on an unsuitable solution or even not converging at all, the function must be appropriately chosen and must correlate with the goals of optimization. Additionally, this function must be rapidly calculated, because the calculation must be repeated many times in order to generate useful results for nontrivial problems. Therefore, the speed of calculation is very important [3]. The priority K(u) of the state u in the priority queue is a two-element vector: K(u) = [K1(u); K2(u)]. For basic application, K1(u) = rhs(u) and K2(u) = g(u) can be used. However, for better performance, more appropriate priority calculations can be chosen.

The method of calculating priorities K(u)—especially in the context of makespan—has a significant impact on the result. Making the appropriate choice in priority calculation will improve the result, and choosing inappropriately will impair the result. This further improved the performance of our algorithm.

Pseudo-code excerpts from our proposed algorithm are shown in Algorithms 1. The first difference is that the optimization of logistic problems may have more than one initial state, which is represented in {04–05}. Additionally, this type of optimization problem does not usually have a defined target state, represented by the while loop removal in ComputePlan(). Main() now continuously calls ComputePlan(), and therefore the algorithm tries continuously to improve the solution. However, we can easily add a new termination condition by replacing the forever loop in Main() by a while/until loop that terminates the optimization algorithm when a predetermined amount of time has elapsed, or after a certain number of repetitions of the procedure [2]. Also, the new state of the parameter *ssol* was introduced, see line {02} which represents the best-found state {13} and contains the current solution of the given optimization problem.

Algorithms 1: The pseudo-codes use the following functions to manage the priority queue: U.Insert(s, k) inserts state s into priority queue U with priority k. U.Remove(s) removes state s from priority queue U. U.Update(s, k) sets the priority of state s in the priority queue to k. U.Remove(s) removes state s from priority queue U. U.Pop() deletes the state with the smallest priority in priority queue U and returns the state. Insert(s, k) inserts state s into priority queue U with priority k. The predicate NotYet(s) is shorthand for "state s has not been expanded yet as overconsistent during the current call to ComputePlan()." Finally StateEvaluation() returns the value of optimized logistic problem.

```
procedure Initialize()
{01} U = Ø;
{02} rhs(s_sol) = g(s_sol) = ∞;
{03} for all s ∈ S rhs(s) = g(s) = ∞;
{04} for all s_start ∈ Starting vertexes
{05}      rhs(s_start) = StateEvaluation();
{06}      UpdateState(s_start);

procedure UpdateState(u)
{07} if (u ≠ s_start) rhs(u) = StateEvaluation();
{08} if (u ∈ U and g(u) ≠ rhs(u)) U.Update(u,K(u));
{09} else if (u ∈U and g(u) = rhs(s)) U.Remove(u);
{10} else if (u ∉ U and g(u) ≠ rhs(u) and NotYet(u)) U.Insert(u,K(u));

procedure ComputePlan()
{11}   u = U.Pop();
{12}     if (g(u) > rhs(u))
{13}              if (rhs(u) < rhs(s_sol)) s_sol = u;
{14}          g(u) = rhs(u);
{15}          for all s ∈ Succ(u) UpdateState(s);
{16}     else
{17}          g(u) = ∞;
{18}          for all s ∈ Succ(u) ∪ {u} UpdateState(s);
```

4. Computational Result

The algorithms were implemented in C# language on a personal computer with Intel Core P7550.

For these experiments, a variation of operation-based representation was chosen, wherein all operations for a job are named by the same symbol in the sequence and interpreted according to the order of occurrence in the given sequence. Therefore, each job appears in the sequence exactly m times and each repetition refers to a unique operation, which is dependent on the context and does not indicate a predetermined operation of the job. This way, any permutation of elements in a sequence always yields a feasible schedule [33].

Although the values of newly acquired states may no longer depend on their predecessors, the position of symbols in the sequence of the newly acquired states does. As for this problem, we changed the symbols position in the sequence only if it was not changed in one of their predecessors.

Experiments performed by means of the graph-based algorithm used three different methods of calculation of priority K(u). The results of experiments marked α were obtained by using the basic calculation of priority K(u): K1(u) = rhs(u) and K2(u) = g(u). For β, priority K(u) consisted of K1(u) = h(u) and K2(u) = rhs(u). Results for experiments marked γ used K1(u) = h(u) and K2(u) = $-$rhs(u), where rhs(u) represents the makespan and h(u) is the sum of completion times of all machines, calculated by the function StateEvaluation() as a byproduct.

Experiments solved by the genetic algorithm were performed using roulette wheel selection, one-point crossover with the corresponding repair process and reciprocal exchange mutation. All tests used the same crossover rate (0.8) and population size (100). The mutation used was relatively low (1/n), where n represents the number of genomes in a chromosome. These parameters were chosen because they showed the best results for this problem type.

Instances of problems for the job shop scheduling problem were commonly used as benchmarking instances [34] and were taken from the database of scheduling problems, OR-Library [35]. One hundred tests on each instance were performed for each algorithm. For a better variation of results, starting states of the graph algorithm were randomly generated.

These experiments were compared by several factors: the time taken to find the best solution, makespan, the average value of makespans found and the average time taken to find these makespans. The results of these tests are summarized in Table 1, which also lists number of jobs, number of machines, instances [36], the best-known solution and the overall time of optimization for the given problem.

We did not test the problem of limiting the proposed algorithm with respect to the number of jobs or machines. Limiting the algorithm by comparison through the most well-known makespans was sufficient enough for the given application area. An example of Gantt graphs for the instance la09 can be found in Figures 1 and 2.

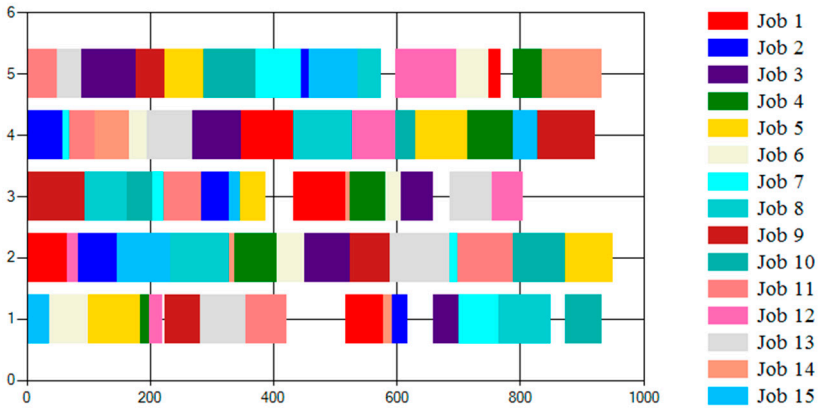

Figure 1. Gantt graph of makespan for la09 as found by graph-based algorithm.

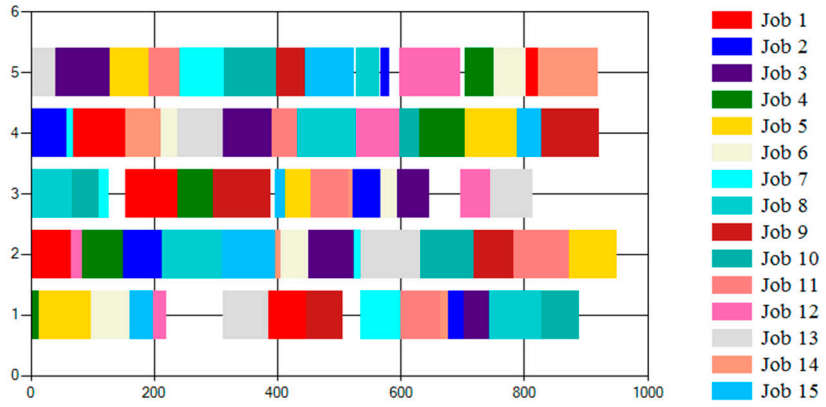

Figure 2. Gantt graph of makespan for la09 as found by genetic algorithm (GA).

The purpose of the performed experiments was not necessarily to reach the best-known makespan, but rather to show what makespan could be found by our proposed algorithm in an allocated time; to how fast it performed and compare it to results reached under the same conditions by an unmodified GA. For this reason, we let each of the tested algorithms run for a certain amount of time—unless the best-known makespan was found.

The results of comparative experiments showed that our proposed graph-based algorithm was able to find the same or better solutions compared to the genetic algorithm—and managed to find them faster. The graph-based algorithm also managed to find the same average makespan as the GA on eight benchmarking instances, and a better average makespan on eighteen instances. However, detailed examination of the experimental

results showed that with an increase in the number of parameters and iterations, the time required for the execution of each iteration will gradually start to rise; under those conditions, the proposed algorithm will perform its iterations slower than the tested genetic algorithm.

Table 1 shows that some of the operations sequences are not similar in order, but are very similar in calculation time. At times, the time to find the best makespan was the same as the time to test various best-known makespans (Table 1, fourth column). We did not discuss this fact in detail, but we do not expect a significant reason.

Despite this fact, the tested graph-based algorithm converged on the optimal solution quickly and obtained better results than the tested genetic algorithm, proving that the behavior shown on several travelling salesman problems in [2,3] can apply to different types of scheduling problems as well.

5. Conclusions

In this article, we proposed a graph-based algorithm that can be used for scheduling-related optimization problems. It was based on the Generalized Framework for Lifelong Planning A*. This algorithm was then tested against the classic genetic algorithm on forty benchmarking instances for the job shop scheduling problem.

The experimental results on the JSSP showed that our proposed algorithm reached the best known makespan in thirteen instances while the genetic algorithm did so only in twelve. The graph-based algorithm found a better solution than the genetic algorithm on twenty-two tested benchmarking instances, found a solution with at least the same makespan as the GA on another twelve instances, and ended in optimality in eleven of these instances. The best-known makespan's average deviation from the best makespan acquired by the graph-based algorithm was 4.7% for α, 5.05% for β and 5.08% for γ; the genetic algorithm had an average deviation of 5.19%, and on all but fourteen instances, the proposed algorithm managed to find the solution faster (on average, about 200–300% faster) than the genetic algorithm. Additionally, on all remaining instances, the graph-based algorithm managed to find a solution in the same time frame as the GA.

The average makespan's average deviation from the best-known makespan was 9.63% for the genetic algorithm, while the graph-based algorithm's variations had an average deviation of 9.22% for α, 10.39% for β and 10.48% for γ. Additionally, on all but six benchmarking instances, the proposed algorithm managed to find a solution faster (on average, about 100%-200% faster) than the genetic algorithm; on five of these instances, the graph-based algorithm managed to find a solution in the same time frame as the GA.

Therefore, we can state that our proposed algorithm generally gives the same or better results—in comparison with the genetic algorithm's basic implementation—on the job shop scheduling problem. The practical consequences of the identified improvements (in makespan and calculation speed) are economic; these improvements will bring a price reduction.

The manner of priority calculation has a nonnegligible impact on solutions. An appropriately chosen priority calculation can improve results, and an inappropriately chosen priority calculation can worsen them. Therefore, the proposed algorithm's performance can be further improved by proposing new methods of priority calculation.

Author Contributions: Conceptualization, J.S. and V.S.; Methodology, J.S.; Software Z.B. and R.K.; Validation, V.S.; Formal Analysis, J.S. and V.S.; Investigation, J.S.; Resources, V.S.; Data Curation, Z.B.; Writing—Original Draft Preparation, J.S.; Writing—Review & Editing, V.S.; Visualization, R.K.; Supervision, V.S.; Project Administration, J.S.; Funding Acquisition, V.S. All authors have read and agreed to the published version of the manuscript.

Funding: The research described in this paper was financed by a grant from the Ministry of the Interior of the Czech Republic, Program of Security Research, VI20192022135 "Deep hardware detection of network traffic of next generation passive optical network in critical infrastructures".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: This article is based upon grant of the Ministry of the Interior of the Czech Republic, Program of Security Research, VI20192022135, "Deep hardware detection of network traffic of next generation passive optical network in critical infrastructures".

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript.

GA	Genetic Algorithm
GLPA*	Generalized Lifelong Planning A*
JSSP	Job Shop Scheduling Problem
LPA*	Lifelong Planning A*

References

1. Skorpil, V.; Cizek, L.; Stastny, J. Path Optimization by Graph Algorithms. In Proceedings of the International Conference CSCC 2012, Recent Researches in Communication and Computers, Kos island, Greece, 14–17 July 2012; pp. 73–78.
2. Cizek, L.; Stastny, J. Comparison of Genetic Algorithm and Graph-based Algorithm for the TSP. In Proceedings of the MENDEL 2013, 19th International Conference on Soft Computing, Brno, Czech Republic, 14–18 July 2013; pp. 433–438.
3. Stastny, J.; Skorpil, V.; Cizek, L. Traveling Salesman Problem Optimization by Means of Graph-Based Algorithm. In Proceedings of the 39 th International Conference on Telecommunications and Signal Processing (TSP), Vienna, Austria, 27–29 June 2016; p. 207.
4. Chen, J.; Zhang, S.; Gao, Z.; Yang, L. Feature-based initial population generation for the optimization of job shop problems. *J. Zhejiang Univ. Sci. C* **2010**, *11*, 767–777. [CrossRef]
5. Sun, L.; Cheng, X.; Liang, Y. Solving Job Shop Scheduling Problem Using Genetic Algorithm with Penalty Function. *Int. J. Intell. Inf. Process.* **2010**, *1*, 65–77.
6. Liu, B.; Wang, L.; Jin, Y.H. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Comput. Oper. Res.* **2008**, *35*, 2791–2806. [CrossRef]
7. Tasgetiren, M.F.; Liang, Y.C.; Sevkli, M. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur. J. Oper. Res.* **2007**, *177*, 1930–1947. [CrossRef]
8. Ge, H.W.; Sun, L.; Liang, Y.C.; Qian, F. An effective PSO-and-AIS-based hybrid intelligent algorithm for job-shop scheduling. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2008**, *38*, 358–368.
9. Coello, C.A.C.; Rivera, D.C.; Cortes, N.C. Use of an artificial immune system for job shop scheduling. *Lect. Notes Comput. Sci.* **2003**, *2787*, 1–10.
10. Yang, J.H.; Sun, L.; Lee, H.P.; Qian, Y.; Liang, Y.C. Clonal selection based memetic algorithm for job shop scheduling problems. *J. Bionic Eng.* **2008**, *5*, 111–119. [CrossRef]
11. Zhang, C.Y.; Li, P.G.; Rao, Y.Q. A very fast TS/SA algorithm for the job shop scheduling problem. *Comput. Res.* **2008**, *35*, 82–294. [CrossRef]
12. Nowicki, E.; Smutnicki, C. A fast taboo search algorithm for the job shop scheduling problem. *Manag. Sci.* **1996**, *41*, 113–125.
13. Dell, A.M.; Trubian, M. Applying tabu-search to job shop scheduling problem. *Ann. Oper. Res.* **1993**, *41*, 231–252. [CrossRef]
14. Wang, T.Y.; Wu, K.B. A revised simulated annealing algorithm for obtaining the minimum total tardiness in job shop scheduling problems Operations. *Int. J. Syst. Sci.* **2000**, *31*, 537–542. [CrossRef]
15. Kolonko, M. Some new results on simulated annealing applied to the job shop scheduling problem. *Eur. J. Oper. Res.* **1999**, *113*, 123–136. [CrossRef]
16. Weng, M.X.; Ren, H.Y. An efficient priority rule for scheduling job shops to minimize mean tardiness. *IIE Trans.* **2006**, *38*, 789–795. [CrossRef]
17. Canbolat, Y.B.; Gundogar, E. Fuzzy priority rule for job shop scheduling. *J. Intell. Manuf.* **2004**, *15*, 527–533. [CrossRef]
18. Klein, R. Bidirectional planning: Improving priority rule-based heuristic for scheduling resource-constrained projects. *Eur. J. Oper. Res.* **2000**, *127*, 619–638. [CrossRef]
19. Gao, J.; Gen, M.; Sun, L.Y. A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Comput. Ind. Eng.* **2007**, *53*, 149–162. [CrossRef]
20. Zhao, L.H.; Deng, F.Q. A hybrid shifting bottleneck algorithm for the job shop scheduling problem. *Dyn. Contin. Discret. Impulsive Syst. Ser. A-Math. Anal. Part 3* **2006**, *13*, 1069–1073.
21. Pezzella, F.; Merelli, E. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *Eur. J. Oper. Res.* **2000**, *120*, 297–310. [CrossRef]

22. Baptiste, P.; Flamini, M.; Sourd, F. Lagrangian bounds for just-in-time job shop scheduling. *Comput. Oper. Res.* **2008**, *35*, 906–915. [CrossRef]
23. Chen, H.X.; Luh, P.B. An alternative framework to Lagrangian relaxation approach for job shop scheduling. *Eur. J. Oper. Res.* **2003**, *149*, 499–512. [CrossRef]
24. Kaskavelis, C.A.; Caramanis, M.C. Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IIE Trans.* **1998**, *30*, 1085–1097. [CrossRef]
25. Brucker, P.; Jurisch, B.; Sievers, B. A branch and bound algorithm for job shop scheduling problem. *Discret. Appl. Math* **1994**, *49*, 105–127. [CrossRef]
26. Artigues, C.; Feillet, D. A branch and bound method for the job shop problem with sequence dependent setup times. *Ann. Oper. Res.* **2008**, *159*, 135–159. [CrossRef]
27. Lorigeon, T. A dynamic programming algorithm for scheduling jobs in a two-machine open shop with an availability constraint. *J. Oper. Res. Soc.* **2002**, *53*, 1239–1246. [CrossRef]
28. Potts, C.N.; Van Wassonhove, L.N. Dynamic programming and decomposition approaches for the single machine total tardiness problem. *Eur. J. Oper. Res.* **1987**, *32*, 405–414. [CrossRef]
29. Kofjac, D.; Knaflic, A.; Kljajic, M. Development of a Web Application for Dynamic Production Scheduling in Small and Medium Enterprises. *Organizacija* **2010**, *43*, 125–135. [CrossRef]
30. Koenig, S.; Likhachev, M.; Furcy, D. Lifelong Planning A*. *Artif. Intell. J.* **2004**, *155*, 93–146. [CrossRef]
31. Koenig, S.; Likhachev, M.; Furcy, D. Speeding up the Parti-Game Algorithm. In Proceedings of the The Neural Information Processing Systems (NIPS), Whistler, BC, Canada, 9–14 December 2002; pp. 1563–1570.
32. Likhachev, M.; Koenig, S. A Generalized Framework for Lifelong Planning A* Search. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, USA, 5–10 June 2005; pp. 99–108.
33. Seda, M. Mathematical Models of Flow Shop and Job Shop Scheduling Problems. *Int. J. Appl. Math. Comput. Sci.* **2007**, *4*, 241–246.
34. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
35. Beasley, J.E. OR-Library: Distributing test problems by electronic mail. *J. Oper. Res. Soc.* **1990**, *41*, 1069–1072. [CrossRef]
36. Lawrence, S. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*; Technical Report, GSIA; Carnegie Mellon University: Pittsburgh, PA, USA, 1984.