Annales Mathematicae et Informaticae
Accepted manuscript
DOI: 10.33039/ami.2024.02.006
URL: https://ami.uni-eszterhazy.hu

Testing the selection heuristic of the Accelerated Branch and Bound method

Emília Heinc[ab], **Balázs Bánhelyi**[ab]

[a]University of Szeged, Institute of Informatics
heincze,banhelyi@inf.u-szeged.hu

[b]University of Győr, Vehicle Industry Research Center

Abstract. This article examined the issue of selection heuristics for the ABB algorithm, a branch-and-bound method for determining the optimal solution structure in P-graphs. Previous studies have not investigated the possible effects of different heuristics on the running time of the ABB algorithm. In this study, we represent the results of applying three basic heuristics in randomly generated P-graphs. In particular, for P-graphs with matrix patterns, the LIFO heuristic is recommended because it performed the best, while the FIFO heuristic had the slowest running time.

Keywords: Branch and Bound, mixed integer programming, production models

AMS Subject Classification: 90C57

1. Introduction

Process network synthesis (or PNS) is a basic tool developed in the 1990s for chemical process problems [14]. Process synthesis aims to find the optimal sub-structure and configuration of an extensive system of functional units and materials [13]. The method is applied in many fields, such as supply chain optimization [12], energy optimization[1], and vehicle scheduling [3, 6]. The search for the optimal sub-network can be formulated as a MILP problem, where the number of binary variables equals the number of units, meaning the complexity of the problem is NP-hard.

The algorithm that finds the optimal sub-network is an accelerated branch-and-bound method (ABB). This method is very similar to the branch-and-bound method for MILP problems. Dakin [5] proposed depth-first search for MILP. This node selection rule always selects a node from the leaf queue with the maximum

Submitted: July 17, 2023
Accepted: February 6, 2024
Published online: February 21, 2024

depth in its search tree. Depth-first search is the preferred strategy for feasibility-only problems. Another technique is Breadth-first search algorithm. It starts at the root of the tree and examines all nodes at the current depth before moving to nodes at the next depth level. This technique is often used when the algorithm needs to parallelize [2]. Besides these two basic algorithms, there are of course other algorithms when additional information is available, such as a heuristic [11]. There are several summary studies of these algorithms in the literature [4], but nowadays more and more algorithms are appearing that use AI for decision making [15].

The ABB algorithm uses the axioms of synthesis to speed up the running of the algorithm. In previous studies, no heuristic for processing order of the non-closed branches to reduce running time was considered. In this work, we tried the simplest heuristics to create an order between the non-closed branches. We tried the following heuristics to determine the order of the non-closed branches: LIFO, FIFO, and random order. We ran the algorithm over randomly generated examples and compared the results based on the number of times the LP solver was called and the runtime.

2. Definitions

2.1. P-Graphs and feasible solutions

P-Graphs. P-Graph is a basic tool for determining the optimal sub-solution structure and configuration of partial solutions for large systems. The main strength of the method is that it combines combinatorial and graph-theoretic techniques.

P-Graphs consist of pairs (M, O), where M is a finite set of materials, and $O(\in \wp(M) \times \wp(M))$ is a finite set of operating units. The graphs can be described as directed bipartite graphs. The set of vertices are O and M sets, and the directed edges represent the connection between operating units and materials.

Process Network Synthesis (PNS) problems are defined by (P, R, O) triplets, where O is similar to set of operating units previously defined in Subsection 2.1. The following will be satisfied in Process In process network synthesis problems, all materials in M can be classified into three categories: P, products, R, raw materials, and I intermediate materials.

Combinatorially feasible solution. A combinatorially feasible solution or solution structure is a special P-Graph, that is made by the materials and operating units from a process network synthesis problem. A P-Graph(M, O) is a combinatorially feasible solution if and only if it satisfies the following five axioms:

1. All materials from P are in the graph.

2. None of the operating units in the graph produces any materials from R.

3. All operating units represented in the graph are from the set O.

4. Every operating unit in the graph has at least one path leading to a material that belongs to the set P.

5. Every material from set M is either produced or consumed by at least one operating unit belonging to O.

2.2. Basic notations linked to operating units and materials

An operating unit is defined by (α, β), where $\alpha, \beta \in \wp(M)$. Set of consumed materials is defined by $X \subseteq O$ denoted by $\mathrm{mat}^{\mathrm{in}}(X) = \bigcup_{(\alpha,\beta)\in o} \alpha$. Set of produced materials is $\mathrm{mat}^{\mathrm{out}}(X) = \bigcup_{(\alpha,\beta)\in o} \beta$. In both cases, the $X \subseteq O$. An operating units in the configuration execute operations and during one operation, a predefined constant amount of materials is consumed and produced. The operating unit $o \in O$ has a cost value, which is defined by the formula, $\mathrm{cost}(o) = \mathrm{fix\_cost} * y + \mathrm{operational\_cost} * x$, i.e. the weighted sum of $y \in \{0, 1\}$, which is set to 1 if the operating unit o is installed in the P-Graph, and $x \in \mathbb{R}_{\geq 0}$, which defines how many operations are executed. Besides the cost of operating units, the raw materials may also have a cost. It is the cost to buy them if they are not available by default.

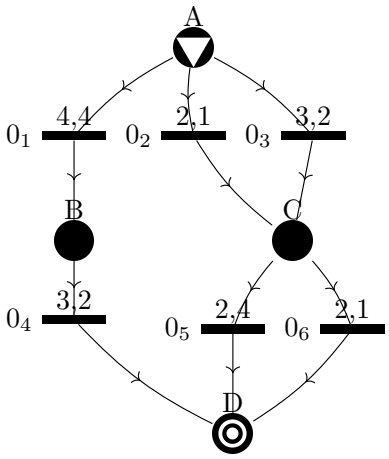
2.3. Decision mappings

It was shown that P-Graphs are basic tools to define the sub-structure of a PNS problem. Another tool, so-called decision mapping[9] equivalent to the P-Graph, can describe sub-substructures. Let $\Delta(m) = \{(\alpha, \beta) \mid (\alpha, \beta) \in O \text{ and } m \in \beta\}$. Let $\delta(m)$ be a subset of $\Delta(m)$, where $m \in M$. δ is also extended to sets, $\delta[m] = \{(X, \delta(X)) \mid X \in m\}$. The complement of decision mapping $\delta(m)$ is $\overline{\delta}(m) = \Delta(m) \setminus \delta(m)$, if $m \in M$. Included operating unit set in the decision mapping $\delta[m]$ will be $\mathrm{op}(\delta[m]) = \bigcup_{X\in m} \delta(X)$. The other notation of the included operating units of decision mapping $\delta[m]$ is O_I. The set of excluded operating units is denoted by $O_E = \mathrm{op}(\overline{\delta}[m])$.

3. Mathematical model of solution structure

Among possible solution structures of a PNS problem, the solution structure that contains all other possible solution structures of the problem is called the maximal solution structure. The algorithm that finds the maximal solution structure is called MSG (Maximal Solution structure Generation) algorithm [7]. Finding a solution structure with minimal total cost for the PNS problem is equivalent to finding an optimal sub-solution structure of the maximal solution structure of the problem. The cost function is composed of the cost of operating units and materials. As mentioned earlier, finding a sub-solution structure forms a MILP problem[10]. The variables come from the operating units. An example of the MILP transformation is shown in Figure 1. The graph in the figure on the left is the maximal structure of the problem. In the object function, the summed cost is

minimized. The first six inequalities define that x is 0 if an operating unit is not installed. B and C are intermediate materials (shown as black circles). Material balance conditions should be defined for intermediate products. These conditions state that at least as much of each intermediate product must be produced as is necessary for the operation of the operating units that use it, otherwise the manufacturing process would come to a standstill. A is the raw material. The aim is to produce the products from the raw materials.



Minimize obj:
$4y_1 + 2y_2 + 3y_3 + 3y_4 + 2y_5 + 2y_6 +$
$4x_1 + 1x_2 + 2x_3 + 3x_4 + 4x_5 + 1x_6$
Subject to:
O_1: $x_1 \leq 1000y_1$
...
O_6: $x_6 \leq 1000y_6$
B: $x_1 - x_4 \geq 0$
C: $x_2 + x_3 - x_5 - x_6 \geq 0$
$x_1, x_2, ..., x_6 \geq 0$
General
$x_1, x_2, ..., x_6$
Binary
$y_1, y_2, ..., y_6$

Figure 1. Transform the maximal structure into a MILP mathematical model.

4. Accelerated Branch and Bound (ABB) method

As it was mentioned in [8], the problem of finding the optimal sub-solution structure is NP-hard. To find the optimal solution, a branch-and-bound technique is used. The algorithm is given in Algorithm 1. The input of the algorithm is the set of materials M and the *PNS* problem whose optimal partial solution structure is to be found, as well as the *datastructure* used in the branching method to keep an ordering between the unsolved branches.

The *ABBD* sub-method is called for each node of the branch-and-bound tree. The parameters of the function are as follows. The first parameter is the set of materials to be produced. Initially, it is set to the set of products. The second parameter is the materials that were already produced. The third parameter is the current decision mapping. The materials that currently need to be produced are in the set p.

The halting condition of the algorithm is when there are no more materials to

produce. A lower bound for the optimal value is calculated for each branch since the value should be minimized. In each branch, a decision is made as to which operating unit set will produce material from p, i.e., a new entry is added to the current decision mapping. When the branching strategy runs, different strategies can be applied as to which node should be extracted next. The strategy is determined by the *datastructure*. The *datastructure* stores elements of p in a predefined order. The *GetElement*() method returns a material, and the sub-branches of the current branch are all possible decision mappings where the material x is produced by a set of operating units. The set of operating units is denoted by c in the algorithm, and it is checked in line 16 that c is consistent with current decision mapping.

The material set p' is the previous state of the materials to be produced, and p will be the state of the new materials to be produced in the sub-branch. It is input materials of the included operating units in the decision mapping, as to operate a unit the input materials have to be produced (except raw materials). The raw materials and currently produced materials (m') do not need to be produced. New materials to be produced ($p \setminus p'$) are added to the data structure, and materials newly left out from p are erased.

The ABBD sub-method is recursively called then for the sub-branch with the modified parameters.

Algorithm 1 ABB algorithm

Input $M, PNS(P, R, O), datastructure$
Global variables $R, \Delta(x), (x \in M), U, currentbest$

1: $U := \infty; currentbest := \infty$
2: $O := MSG(PNS(P, R, O))$
3: ABBD(P, $\emptyset$, $\delta[\emptyset]$)
4: **return**
5: **end procedure**

1: **procedure** ABBD(p, m, $\delta[m]$)
2: $bound = Lower\_Bound(PNS(P, R, 0), O_I, O_E)$
3: **if** $p = \emptyset$ **then** *Halting condition.*
4: **if** $U \geq bound$ **then**
5: $U = bound$;
6: update currentbest;
7: **end if**
8: **return**
9: **end if**
10: **if** $bound \geq U$ **then** *Cutting the branch.*
11: **return**
12: **end if**
13: $x := datastructure.GetElement()$;
14: $C := \wp(\Delta(x)) \setminus \{\emptyset\}$;
15: **for** $\forall c \in C$ **do**

```
    if ∀y ∈ m, c ∩ δ̄(y) = ∅&(Δ(x) \ c) ∩ δ(y) = ∅ then
      m' := m ∪ {x};
      if S(δ[m']) = ∅ then
        Continue;
      end if
      δ[m'] := δ[m] ∪ {(x, c)};
      p' := p;
      p := (mat^in(op(δ[m'])) ∪ P) \ (m' ∪ R);
      datastructure.Insert(p \ p');
      datastructure.Erase(p' \ p);
      O_I := op(δ[m']);
      O_E := op(δ̄[m']);
      ABBD(PNS(P, R, O), p, m', δ[m'])
    end if
  end for
end for
return
end procedure
```

4.1. Heuristics

When selecting a new material to process, various strategies can be used to expand the current decision mapping. In previous studies, nodes were selected according to the alphabetical order of materials. It was not examined, how the selection methods affect the total running time of the ABB algorithm. The following strategies were tried: FIFO, LIFO, and Random Pick (RND). During the LIFO, the algorithm works as a DFS (Depth-first search) algorithm, and for the FIFO case, it is a BFS (Breadth-first search) algorithm.

Consider the P-Graph shown in Figure 2 and assume that no heuristics is specified for the traversal of the graph. The default values are applied when there is no specified amount of materials produced and consumed.

The simplest way is to order the nodes in alphabetical order of the names. In this case, the G, H will be the first materials to produce the p. If G is first produced then there are two ways to produce G by the path of O_3, O_4 or by O_1, O_4. According to the ABB algorithm, in this scenario, the path O_3, O_4 is examined first, and then all possible steps are executed to produce material H. It took extra five steps to find out. The same is repeated for the other two additional branches of producing G, the cases O_4, O_1 and O_4, O_1, O_3. The reason is that branch in both cases cannot be cut by the relaxed lower bound. This is because the upper bound is 12 since all fixed and default costs of operating units are 1. If O_3 and also O_1 produce D, then the relaxed bound is calculated from the costs of the included operating units added to the operating costs of the free operating units. The number of steps evaluated to 15. If in the root node, when deciding whether to produce G or H first, it is decided that H should be produced first, then the algorithm would stop running after 11 steps. If the decision in the first step were completely random

in which orders the products, then the expected value of the run in the case of alphabetical order would be $\frac{1}{2} \cdot 17 + \frac{1}{2} \cdot 11 = 14$.

If the graph is evaluated according to the heuristic of LIFO, the last added material is always processed. In the original case, this is product H. In this heuristic, the algorithm finds the shortest chain of materials and operating units that leads to the raw material A. In the example, this takes five steps. After that, G is considered. If D is produced from raw material B, the process takes two additional steps, and the same is true if the source material is A, and if D is produced by both O_3 and O_4, one additional step is needed to examine this branch. This results in a total of 9 steps to find the optimal solution. This is the best-case scenario. If H and G are swapped, then the same worst-case scenario is executed when the algorithm is run in the order G, H using the alphabetic heuristic. The result is the same running time 17. Overall, the expected value of the LIFO heuristic for this example is $\frac{9+17}{2} = 13$.

In the case FIFO, if the order of processing is G, H first, then after G is produced by O_4, H is examined. Then H is produced by O_7, and after that D will be the selected material. The production of D has three branches, and these branches have the same additional four steps to reach the raw material A. So, in total, there are 15 steps. In the case of H, G, first H is produced by O_7, then G is examined and produced by O_4. After that, the F is produced by O_6, and then the branching is done how to produce the material D. A total of four steps are used to examine how D can be produced. After the three branches were examined, it took additional three steps to be processed. This means that in this case, the algorithm would stop running after 13 steps. The expectation value of the FIFO case heuristic will be $\frac{15+13}{2} = 14$.

This simple example shows that the LIFO strategy works the fastest on average (13 steps), compared to the random and FIFO algorithms (14 steps). Of course, this strategy also depends heavily on which product of the same step in the ABB algorithm is processed first. There are two cases, one with 9 and the other with 17 steps.

5. Results

The algorithm was also tried on randomly generated examples. The operating units and materials are ordered in a matrix pattern. The graph has a *height* number of layers and in each layer, there is a *width* number of operating units or neighbour materials. The first layer consists only of raw materials, and the last layer contains only product materials. The type of neighbor layers alternates between material and operating unit. Connections are made only between two consecutive layers. Every material in the i-th layer except raw materials is produced by at least one operating unit from the $(i-1)$-th layer. Accordingly, they are also consumed by at least one operating unit from the $(i+1)$-th layer. In addition to these connections, other connections with random p-value $(0 \geq p \geq 1)$ are randomly produced between two layers. All connections are generated by a uniform random distribution. The

graph generation pattern can be seen in Figure 4. In our experiment, the width and height were set to 5, and p was set to 0.2.

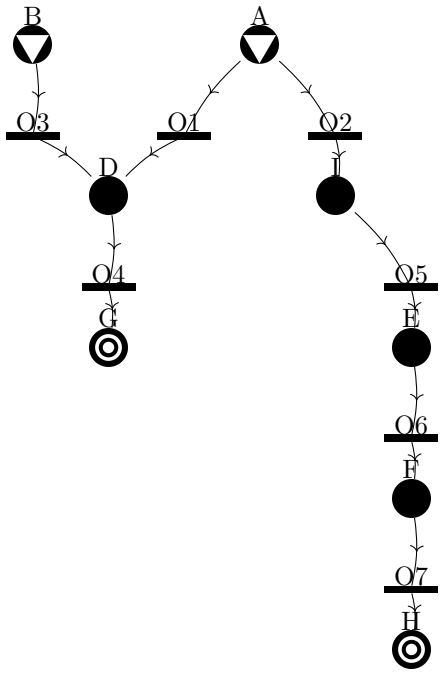


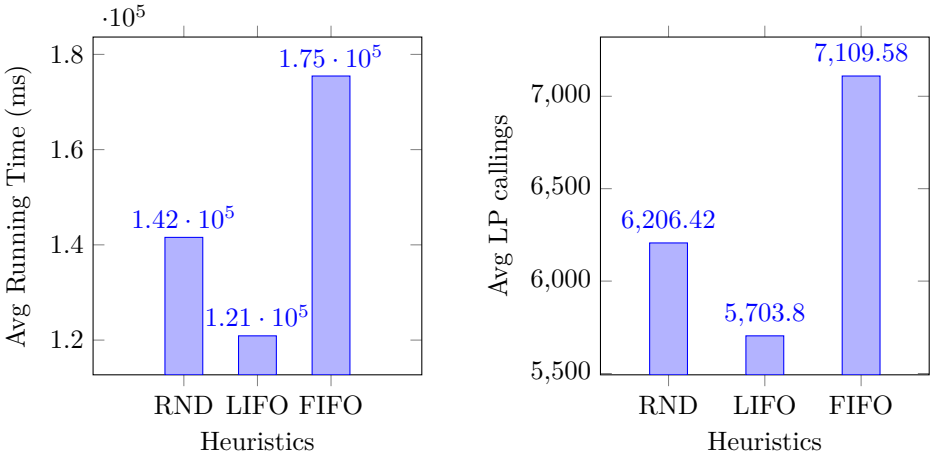
Figure 2. Example P-Graph for different heuristics.

100 examples were generated uniformly at random and the ABB algorithm was run with three different heuristic settings. The random selection heuristic was solved so that the nodes were indexed in random order. In the node selection part, the elements of the materials to be produced were ordered by increasing indexes. The results are grouped by heuristics and the average running time values are collected. Two types of running times were measured. The first is the CPU per clock per second multiplied by 1000. The second is the number of LP solver callings. The lower bound sub-method is used to calculate the relaxation of the MILP transform associated with the current sub-problem. It is particularly relevant to consider the number of solver callings, as well, since the operating units can differ from the simple linear transition between the produced and consumed materials, i.e. they may also be nonlinear or stochastic.

Running time was calculated as CPU clock per second multiplied by 1000. For these randomly generated graphs, the LIFO algorithm performed the best. The FIFO algorithm was the slowest, and the random pick over-performed it as well.

The random pick heuristic itself can be interpreted as choosing the nodes in the graph by alphabetical order where randomly assigning the name of the materials. In the previous cases, generally alphabetical order was used for every case. The

present paper shows that it is worth reconciling the node selection heuristic.



(a) Average running time of heuristics.

(b) Average number of LP callings of heuristics.

Figure 3. Comparison of different heuristics.

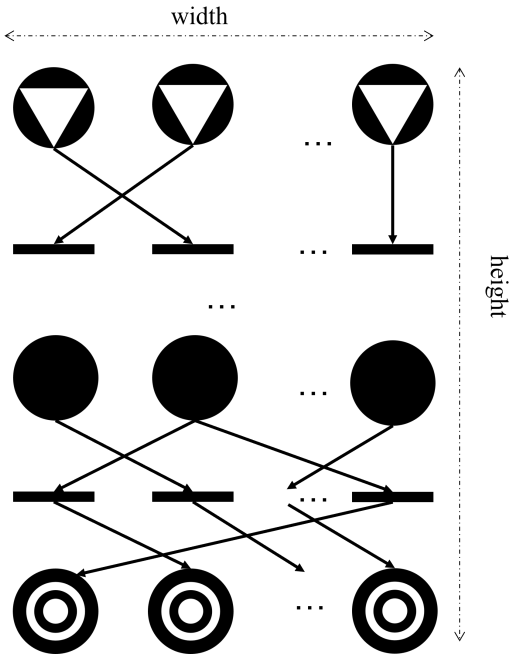


Figure 4. The basic structure of the examined p-graphs.

6. Conclusion and future work

In this article, we have presented the algorithm ABB, which is used in the context of the P-graph. This is very similar to a branch-and-bound method used for standard binary LP problems. We have studied the effect of different traversals of the branches on the running time and the number of LP problems to be solved. In the paper we studied only the 3 best known and simplest versions, but this also shows how differences arise on general graphs. From these results, it can be concluded that it is worth trying other algorithms that are even more computationally demanding. These selection heuristics can be used to achieve even further optimizations that also improve runtime. Such algorithms can be especially interesting when the evaluation of the optimization models is even more time-consuming, for example in nonlinear or stochastic cases.

In the future, we plan to investigate more sophisticated heuristics that use the cost functions and the structural nature of the possible solution structures or other factors.

Acknowledgements. The research presented in this paper was funded by the National Laboratories 2020 Program – Artificial Intelligence Subprogram – Establishment of the "National Artificial Intelligence Laboratory (MILAB) at Széchenyi István University (NKFIH-870-21/2020)" project.

References

[1] K. B. Aviso, J.-Y. Lee, J. C. Dulatre, V. R. Madria, J. Okusa, R. R. Tan: *A P-graph model for multi-period optimization of sustainable energy systems*, Journal of Cleaner Production 161 (2017), pp. 1338–1351, ISSN: 0959-6526, DOI: 10.1016/j.jclepro.2017.06.044.

[2] Z. Bagóczki, B. Bánhelyi: *A parallel interval arithmetic-based reliable computing method on a GPU*, Acta Cybernetica 23.2 (Jan. 2017), pp. 491–501, DOI: 10.14232/actacyb.23.2.2017.4.

[3] M. Barany, B. Bertók, Z. Kovács, F. Friedler, L. T. Fan: *Solving vehicle assignment problems by process-network synthesis to minimize cost and environmental impact of transportation*, Clean Technologies and Environmental Policy 13 (2011), pp. 637–642, DOI: 10.1007/s10098-011-0348-2.

[4] T. Berthold: *Primal Heuristics for Mixed Integer Programs*, in: 2006.

[5] R. J. Dakin: *A tree-search algorithm for mixed integer programming problems*, Comput. J. 8 (1965), pp. 250–255, DOI: 10.1093/comjnl/8.3.250.

[6] Z. Ercsey, A. Nagy, J. Tick, Z. Kovács: *Bus Transport Process Networks with Arbitrary Launching Times*, Acta Polytechnica Hungarica 18 (2021), pp. 125–141, DOI: 10.12700/APH.18.4.2021.4.7.

[7] F. Friedler, K. Tarjan, Y. Huang, L. Fan: *Combinatorial algorithms for process synthesis*, Computers & Chemical Engineering 16 (1992), European Symposium on Computer Aided Process Engineering—1, S313–S320, ISSN: 0098-1354, DOI: 10.1016/S0098-1354(09)80037-9.

[8] F. Friedler, J. B. Varga, E. Fehér, L. T. Fan: *Combinatorially Accelerated Branch-and-Bound Method for Solving the MIP Model of Process Network Synthesis*, in: State of the Art in Global Optimization: Computational Methods and Applications, Boston, MA: Springer US, 1996, pp. 609–626, ISBN: 978-1-4613-3437-8, DOI: 10.1007/978-1-4613-3437-8_35.

[9] F. Friedler, J. Varga, L. Fan: *Decision-mapping: A tool for consistent and complete decisions in process synthesis*, Chemical Engineering Science 50.11 (1995), pp. 1755–1768, ISSN: 0009-2509, DOI: 10.1016/0009-2509(95)00034-3.

[10] F. Friedler, Á. Orosz, J. P. Losada: *P-Graphs for Process Systems Engineering, Mathematical Models and Algorithms*, London: Springer, 2022, DOI: 10.1007/978-3-030-92216-0.

[11] P. E. Hart, N. J. Nilsson, B. Raphael: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics 4.2 (1968), pp. 100–107, DOI: 10.1109/TSSC.1968.300136.

[12] H. L. Lam, P. S. Varbanov, J. J. Klemeš: *Optimisation of regional energy supply chains utilising renewables: P-graph approach*, Computers & Chemical Engineering 34.5 (2010), Selected Paper of Symposium ESCAPE 19, June 14-17, 2009, Krakow, Poland, pp. 782–792, ISSN: 0098-1354, DOI: 10.1016/j.compchemeng.2009.11.020.

[13] N. Nishida, G. Stephanopoulos, A. W. Westerberg: *A review of process synthesis*, AIChE Journal 27.3 (1981), pp. 321–351, DOI: 10.1002/aic.690270302.

[14] J. J. Siirola: *Industrial Applications of Chemical Process Synthesis*, Advances in Chemical Engineering 23 (1996), pp. 1–62, ISSN: 0065-2377, DOI: 10.1016/S0065-2377(08)60201-X.

[15] J. Zhang, C. Liu, X. Li, H.-L. Zhen, M. Yuan, Y. Li, J. Yan: *A survey for solving mixed integer programming via machine learning*, Neurocomputing 519 (2023), pp. 205–217, ISSN: 0925-2312, DOI: 10.1016/j.neucom.2022.11.024.