Development of the algorithm for the optimization of objects pick up sequence on a conveyor belt

A V Seredkin1,2,*, M S Bobrov1, M P Tokarev1, and O A Gobyov1

1 Kutateladze Institute of Thermophysics, Siberian Branch of RAS, 1 Lavrentyev Avenue, Novosibirsk, 630090, Russia
2 Novosibirsk State University, 2 Pirogov Street, Novosibirsk, 630090, Russia
*Email: sasha7-93@mail.ru

Abstract. With recent advances in robotics, various methods and systems for robotic movement planning have gained popularity. In particular, the problem of planning of pick and move operations for the robotic sorting cells has drawn certain attention because new strategies and algorithms become a necessity as a robotic sorting systems application area extends. In the present work, a new algorithm for optimization of sorting operation sequences is proposed and results of its testing on a computer simulation are presented. The algorithm is based on the state tree search method and is aimed for performance improvement of the systems in which the input load can exceed the capacity of the robotic manipulator.

1. Introduction
Control systems based on predictive models for the movements planning of robotic devices have gained wide popularity in recent times. A separate class of such problems, considered in the queuing theory [1], is the planning of the pick-up and movement of objects on the conveyor. In some systems, the performance of the robotic cell may not be sufficient to provide pick up and movement of all of the objects during maximum load. In this case, simple scheduling algorithms such as FIFO (first in, first out), SPT (shortest processing time) and SRPT (shortest remaining processing time) [2], [3] are ineffective. To improve the efficiency of such systems, dynamic optimization of the processing operation management is used. Such optimization is not trivial to implement, as the serving time for every object is different and changes in time. Besides, every object can be picked only within a limited timeframe. Some examples of such optimization can be found in [3], in which two methods of improving classical algorithms were presented. Those improvements made it possible to increase the share of picked objects by up to 5%.

In the present work, a new algorithm for planning the pick and move operations for objects on a conveyor belt based on a tree traversal is proposed and tested on computer simulation. The algorithm was the primary one for implementation in the robotic cell for municipal waste sorting, being developed at the IT SB RAS [4]. In such systems, a robotic sorting unit can be paired with an advanced machine vision subsystem and other sensing devices [4], [5] to acquire information about the stream of wastes on the conveyor belt. In this case, further sorting can be reduced to the problems of gripping objects and planning of pick and move operations. The latter is one of those that require dynamic optimization of the operations queue, as the number and characteristics of waste objects in the input stream can significantly vary in time.
2. Formulation of the problem
Let us define the problem formulation. We shall consider the sorting unit environment that contains a mechanical arm, a moving conveyor belt, and several items transported on the belt. Mechanical arm position (coordinates), belt speed and position, value, destination point for each of the items on the belt are known. The arm can be moved around its working area and the time interval required to travel the arm between any two points within the working area is also known. The arm can pick up items when placed directly above them and can drop items at the destination points (collecting bins). Every time the arm drops an item at the destination point the item is considered as "collected". The goal is to maximize the total value of collected items by controlling arm transitions and pick up/drop operations. The algorithm should work in real-time to provide a continuous sorting unit operation.

In a real application, we only have information about several next items which are on the belt, while the information about items that will be loaded in the future is unknown. While the belt is moving, new items are loaded to the conveyor input and the program gets access to that information.

Let us then define a solution as any possible sequence of arm transitions and pick up/drop operations. We can use several assumptions to narrow the range of possible solutions. Let us consider that in every sequence of operations the arm picks up an item from the belt and drops it to the destination point of the item. In such a sequence of operations, the fastest possible trajectory for each maneuver is used. Let us call such sequence a move. Under this assumption, every solution is the sequence of moves. The algorithm should provide a decision for the next move upon finishing a current move.

Let us also define a state as a snapshot (all known information) of the environment at the given moment. State contains:
- The mechanical arm position;
- Current time;
- Belt speed and size (constant during operation);
- Total collected value;
- List of items on the belt with the information about every item that includes coordinates, value, destination point coordinates and unique identifier (index).

Instead of storing the item coordinate \( x \) (along the conveyor) belt, we evaluate and store the time, when the \( x \) coordinate of the item was equal to 0 using the belt speed and the information from the detection system. This allows reusing the items list for multiple states. For any state and any allowed move, we can simulate the resulting state.

It is useful to note that items on the conveyor belt are already ordered along \( x \)-axes. The cases in which multiple items have the same \( x \) coordinate are rare and these items can be treated in any order. That gives one more assumption for the best solution: the items should be collected from the belt in the same order as they are coming. This reduces the problem to the decision on collecting or skipping for each of the items.

3. Algorithm
Let us assume that there are \( N \) known items on the belt. Theoretically, there are \( 2^N \) possible solutions. The average number of moves in those solutions is \( N/2 \), so to test all possible solutions \( N \times 2^{N-1} \) moves have to be simulated. Some of the solutions have a common history (sequence of moves) up to a certain moment. This common sequence has to be simulated only once for all of the solutions, which contain this sequence. This makes it possible to store all solutions in a tree structure, where nodes are states and edges are moves between them. The root of the tree is the initial state, and leaves are final states. That tree contains \( 2^{N-1} \times 2 \) moves, which is still too much for real-time simulation.

Every leaf corresponds to a unique solution, so the problem is to find one with the highest total value collected. In such a formulation, the problem is common for turn-based games. The total collected value is cumulative (cannot decrease), so is the total value of skipped items (loss). We use that fact to sort tree nodes and guide the algorithm.

The algorithm performs an iterative extension of the state tree (see figure 1). The iteration consists of 4 steps: we follow the best solution, while these nodes are contained in the tree (Selection), then we
model a new state (Simulation) and add a new node (Expansion), as the final step we update the information at the nodes along the path visited (Backpropagation). This sequence of operations was chosen by analogy with the Monte Carlo tree search algorithm [6]. To find the best current solution without traversing all the leaves in the tree, a pair is stored in each node: the lost value, and the current time (hereinafter loss). If there are moves from a node that have not yet been modeled, then loss values are assigned from the state of this node. But if all moves from a given node were simulated, then the loss of the node is assigned to the minimum loss among the node children. As noted earlier, with this approach, the value of loss will never decrease. When searching for a child node with minimal loss, its index can be stored in order to use this information to find the best current solution later. In the modeling step, a new state is evaluated by the algorithm using the information about the last state and turn. The modeling step can have two outcomes: a new state is evaluated and a new node is added to the tree, or the move turns out to be invalid if the item leaves the working area before the arm reaches it. After that, the loss is updated for all the predecessors of the new pick.

**Figure 1.** State tree example. At the root of the tree, the first of three items is picked on the conveyor. The various states of the system are shown in rectangles. The circles inside the states indicate the items, their status is shown in color: green – collected, orange – missed, blue – not yet visited. The black arrow indicates the item that is being captured at the moment. The moves are shown as ovals in which the next capture target is written. The arrow of the move indicates the state to which this move leads.

The tree traversal is completed when the current best solution is modeled to a state from which there are no available moves. This means that such a solution has processed all of the items and found a sequence of moves that minimizes the total value of missed items. Adding new items to the belt can change the current solution, so if new items appear in a queue than new iterations of the algorithm are performed. When the time comes to choose the next move, two situations are possible: the tree traversal is either completed or not. In case when the tree traversal is not completed, information about the structure of the tree is used to make a decision on the next move. Namely, the move that contains the largest subtree is selected. Such choice is based on the assumption, that when modeling, the current best solution is always selected, which means that the move with the largest subtree was the best more often
than the alternatives. This does not guarantee an optimal solution but allows one to guide the algorithm, providing mainly “good” decisions in a case of limited time for processing. In the case of finished tree traversal, the first move of the best solution (guaranteed if new items no longer appear in the queue) or follow the same principle as when the traversal is not completed.

After choosing a move, the node into which this move led becomes a new root, and the rest of the tree is discarded. Items that were picked earlier are removed from the list of items. Time and lost value accumulate from the beginning, which makes it possible to use the selected subtree without changes.

4. Simulator and visualization

To study the behavior and optimize the algorithm for collecting items from the conveyor belt in various conditions, a graphical simulator of a sorting unit was implemented. A simulator environment includes a conveyor belt, a generator of items, containers for collecting items of each type, and a manipulator. The generator of items generates an input stream of items with assigned type, value, and initial coordinates. Manipulator takes items from the conveyor belt and carries them to the destination points (collection bins).

The simulator was implemented in C++ programming language using the open freeglut library for visualization. Visualization of the conveyor belt, manipulator, and bins was carried out using standard tools of the freeglut library. Three-dimensional models of each sortable type of items (cardboard, glass, transparent and opaque plastic) were used as items for collection. Three-dimensional models of items were taken from the open resource (free3d.com). The procedure for reading the coordinates of vertices, edges, and their normals from .obj model files and their rendering using the freeglut library was implemented.

The visualization of the system is shown in figure 2. A legend that displays the value of various types of items, as well as the total value of collected items at the moment is placed in the upper left part of the scene. The entire conveyor is presented in the lower-left part of the scene. The area in which random items are generated is highlighted on the conveyor belt with a blue square and is displayed in detail in the upper right part. In the area highlighted by the green square, items are selected by the manipulator and sorted by the appropriate baskets. A detailed image of this area is located in the lower right of the scene. The manipulator is shown in the form of a green marker, and the collection bins are in the form of red boxes. The bins provide information on the number of picked and skipped items of the corresponding type. Summary statistics on the total number and value of picked and skipped items are shown above the conveyor next to the location of bins. The sequence of items to pick according to the current best solution is shown by a green broken line.

![Figure 2. Visualization screen of the sorting unit simulator.](image-url)
To implement a continuous parallel operation of the simulator and the selection algorithm, a multi-threaded scheme was implemented. In the first thread calculation and rendering of the scene takes place, while in the second thread the algorithm for optimizing the collection of items runs continuously. It was also necessary to implement a process for the secure exchange of information between streams without data loss. A regular process of reading the value of a common variable was added to each stream, allowing operations with common data (information about new items on the belt, information about the graph subsequent selections of items from the list) to only one of the streams. Upon completion of the read or write operation in one of the streams, the value of the variable is changed, and access to the shared data is granted for the functions of the other stream.

5. Results and discussion
To calculate the effectiveness of the algorithm, a simulation was performed. In the simulation, four types of items were fed to the input conveyor. Type and initial coordinates were chosen randomly using a uniform distribution. For each type, there was a collection bin on the side of the conveyor belt. The cost of the items was set regardless of their type and was also chosen randomly from the following uniformly distributed options: 5, 7, 10, 14, and 20. The values of simulation parameters, such as the belt speed, mechanical arm movement speed, and dimensions of the conveyor were chosen close to the waste sorting unit in development [4]. The characteristic time for movement from the item to the collection bin and then to the next item was about 2 seconds. The simulation took place in real-time. Items were generated at the conveyor input with an average interval of 1 s so that the arm could collect about half of the items. The maximum time interval for generating a new item at the input was set to two seconds to limit the dispersion of the input rate.

To estimate the new selection algorithm performance, in addition to the developed algorithm a simpler baseline algorithm was tested under the same conditions. For the baseline algorithm we tried to implement rules that are simple and easy to interpret, yet still adequate and effective for the case. A simple “baseline” algorithm, just as the one described in the paper, makes a decision on the next move (select a next item to pick) after the arm drops an item into the bin. The next item to pick was selected according to the following rule: first, all of the items that can be picked and carried to the bin within the next 2 seconds are selected. Among these items the most “expensive” ones are selected, and then the closest to the end of the working area of the arm is selected from the list of items of the highest value. A simple algorithm does not miss expensive items, but at the same time collects items at a good pace. With a decrease of the “maximum time interval between items” parameter, the probability of missing expensive items increases and the collection rate decreases.

Both algorithms were in identical conditions, including the generated sequence of items. The simulation results are presented in figures 3 and 4.
Figure 3. Comparison of the total cost of collected items, the proposed algorithm (blue), and a simple algorithm (orange) for the first 500 s. The difference between the two graphs (gray) is shown on a different scale (right axes).

Figure 4. The final distribution of collected items by cost using the proposed algorithm (blue) and a simple algorithm (orange).

The simulation lasted for more than 508,000 s. According to the simulation results, the algorithm proposed in the article collected 9.5% more items than the simple algorithm and provided an 11.5% advantage in the total cost of the collected items. A proposed algorithm collected almost all of the items of maximum value. According to the statistics, planning made it possible to significantly increase the
number of collected items of high and average value, reduce the average distance of movement, but also reduced the number of collected items of low value.

**Conclusions**
The algorithm for dynamic planning of sorting operations on a conveyor was proposed in the paper. The algorithm is based on the iterative extension and traversal of the state tree, in which the nodes are the states of the environment and edges are the moves that propagate the states. To test an algorithm a simulator of the sorting unit was implemented. The performance of the proposed algorithm was tested on the simulation and compared to the performance of the simpler algorithm. The proposed algorithm has shown a significant advantage in the raw number of collected items, the total cost of the collected items, and the average distance of movement of the manipulator that performs sorting. Although the algorithm was developed mainly to optimize the sorting of solid waste on the conveyor, it can be employed in many areas, in which automated sorting takes place. To ensure the advantages of the newly developed algorithm more tests on the real sorting unit with a robotic manipulator will be performed in the future.

**Acknowledgment**
The work was supported by the Ministry of Science and Higher Education of the Russian Federation (grant Agreement No. 14.604.21.0203, identification number of the project RFMEFI60418X0203).

**References**
[1] Gnedenko B V Kovalenko I N 1989 *Introduction to Queueing Theory, Second Edition* (Boston: Birkhauser)
[2] Krechetov, I., Skvortsov, A., Poselsky, I., Lavrikov, P. Scheduling jobs and managing a robotic sorting node 2018 *Int. J. Mech. Eng. Technol* 9(11) 95–105
[3] Mattone R, Divona M, Wolf A 2000 *Robotics and Computer-Integrated Manufacturing* 16(2–3) 81–90
[4] Seredkin A V, Tokarev M P, Plohih I A, Gobyzov O A, Markovich D M 2019 *JoP: Conf. Series* 1359 012127
[5] Lapusan C, Rad C, Hancu O, Brisan C 2018 *IOP Conf. Series: Materials Science and Engineering* 444 052030
[6] Chaslot G M J B, Winands M H M, Herik H J V D, Uiterwijk J W H M, Bouzy B 2008 *New Mathematics and Natural Computation* 04(03) 343–57