Efficient Robot Routing in Factorio

dexteritas
dex.teritas@gmx.de

September 3, 2023

Abstract

The computer game Factorio[1] contains as an important element flying robots which can be used for transporting objects as well as for constructing buildings. Since thousands of such robots can be on the move at the same time in a game, intelligent routing procedures have not been used so far for performance reasons. In some cases this can lead to unfavorable behavior. In this paper, we present an approach that enables smarter routing of robots while being preformant.

Structure: First, a concrete problem with the current state of the game is presented in section 1. Then, section 2 specifies problems that need to be addressed by in an improved routing procedure. Section 3 briefly describes pathfinding in general and in the specific case. Afterwards, the new intelligent routing approach is introduced in section 4. Finally, section 5 gives a conclusion.

1 Current State

The last section "Mitigation for robot pathing over lakes" of Factorio Friday Facts #374 [kOK23] addresses problem that may arise in larger non-convex roboport networks (see fig. 1). The robots fly the direct path to their destination until the battery runs out, if necessary, and they fly to the nearest Roboport. No pathfinding is done at the beginning and it is not checked if the destination can be reached directly.

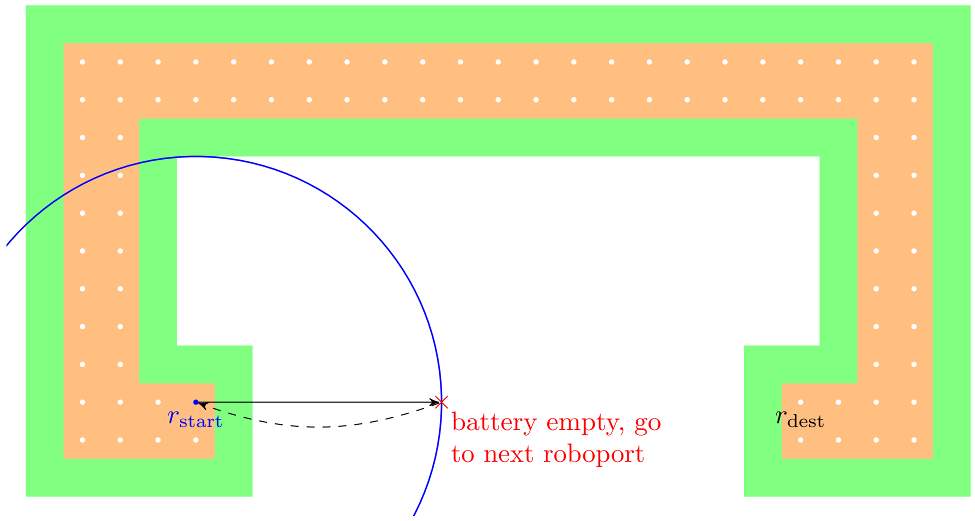

Figure 1: Robot movement v1 – The current approach in factorio 1.1. The small dots are Roboports, the logistics area is shown in orange and the construction area is shown in green, as it is in the game. The robot flies directly to the destination r_{dest}, but on the way it realizes that the battery is empty (red $\times$) and flies to the next Roboport, which in this case is the initial Roboport r_{start}. The slow flight mode is represented by the dashed arrow. The curvature is only for better readability; the direct air line is also taken.

[1] https://factorio.com

The new variant also first tries to fly directly to the destination. Only the behavior when the battery is empty has been changed so that the roboport that is flown to afterwards should be closer to the target than the current robot position (and therefore the previous roboport) in order to avoid endless loops (see fig. 2).

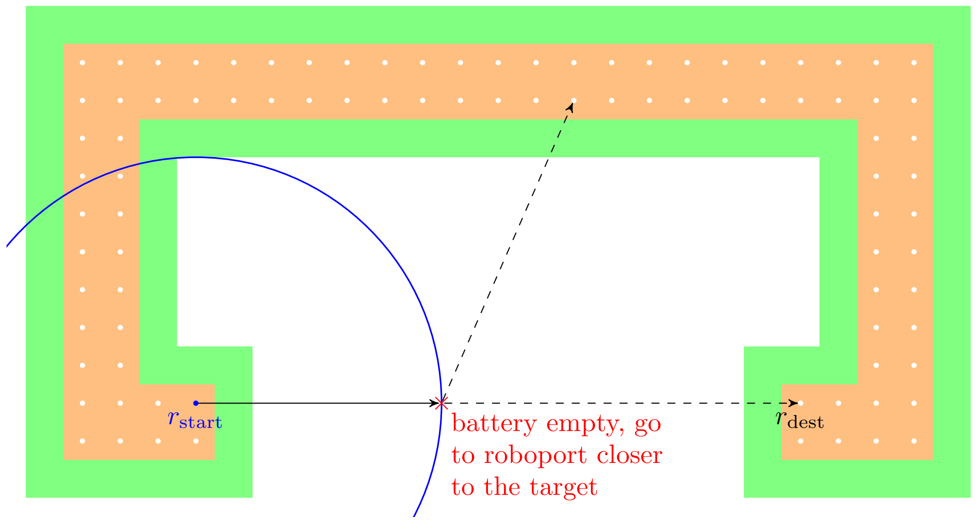

Figure 2: Robot movement v2 – New approach from FFF 374 [kOK23]. The robot flies directly to the destination r_{dest}, but on the way it realizes that the battery is empty (red $\times$) and flies to a roboport, which is closer to the target r_{dest} than the robot is. The slow flight mode is represented by the dashed arrow.

1.1 Runtime Estimation for Different Scenarios

Let us first consider the current approach (Robot movement v1). Currently, a linear movement to the target is started directly without prior checking. Meanwhile, it should be checked every tick whether the battery is empty or not. Whether the battery is actually checked every tick or an event is triggered when it is empty is unknown to me. Algorithm 1 describes the current behavior approximately.

Algorithm 1: Current robot movement

```
Data: robot $b$, goal $g$
$b_{\text{dest}} = g$;
while $b_{pos} \neq b_{dest}$ do
    if $b_{charge} > 0$ then
        move_to($b_{\text{dest}}$);
    else
        $r$ = get_next_roboport($b_{\text{pos}}$);
        move_slowly_to_roboport($r$);
    end
end
```

Now lets estimate the computational effort of the current approach for different scenarios:

S1. The destination can be reached directly without an intermediate loading stop.

S2. The target is out of robot range and a stop must be made.

S3. A Roboport is removed.

In scenario S1 the calculation effort is minimal. Only the check of the loading state must take place regularly or it must be calculated initially once at which place the loading will be exhausted. With S2 the same applies as before, but now additionally the next Roboport must be found (get_next_roboport(b_{pos}). For this, a search within the set of roboports of the current logistics network has to take place and the distance has to be considered – a somewhat more complex calculation. By moving slowly, robots are generally in the air longer, increasing the total number of ticks that must be performed for the current action, which can be slightly disadvantageous. Note that S2 occurs both in non-convex logistic areas like fig. 1, and within a convex logistic area (e.g., a large rectangular structure). If the targeted roboport is removed (S3), the next roboport must be determined again.

2 Problems of Intelligent Robots

There are some problems that must be considered and solved in an intelligent routing procedure:

P1. Pathfinding is CPU expensive and therefore bad for UPS.

P2. Storing a precalculated path needs more memory.

P3. Changes in the logistic networks (roboport added or removed) must be considered and would need extra updates of the path.

3 Pathfinding

Pathfinding for ground units in computer games is generally computationally expensive, since not simply the direct path (as the crow flies) is followed, but obstacles (e.g. cliffs, buildings, other units, etc.) are taken into account. For this purpose, for example, the A* algorithm can be used (possibly with hierarchical optimizations as in FFF #317 [OTK19]). However, it may happen that in the course of the movement obstacles block the planned path and thus a recalculation becomes necessary. Since there are too many ways for units to move on a large map, paths are generally not saved but recalculated by each unit or group of units.

Compared to this, the flying robots can move very freely and often actually simply use the beeline. Only the battery state of the robot and thus its range must be taken into account. Since there are no obstacles as described above for the robots, a recalculation would have to take place only with a change of the Logistic Network (see P3).

4 Intelligent Routing Approach

Intelligent robots would try to avoid the empty battery and thus also a slow movement completely by doing some kind of route planning at the beginning. This flight path to avoid empty battery is shown in fig. 3. The circles here represent the maximum range of the robot type currently under consideration. The color-coded Roboports are used for charging. The result is the path from Roboport r_{start} to r_{dest} drawn with arrows.

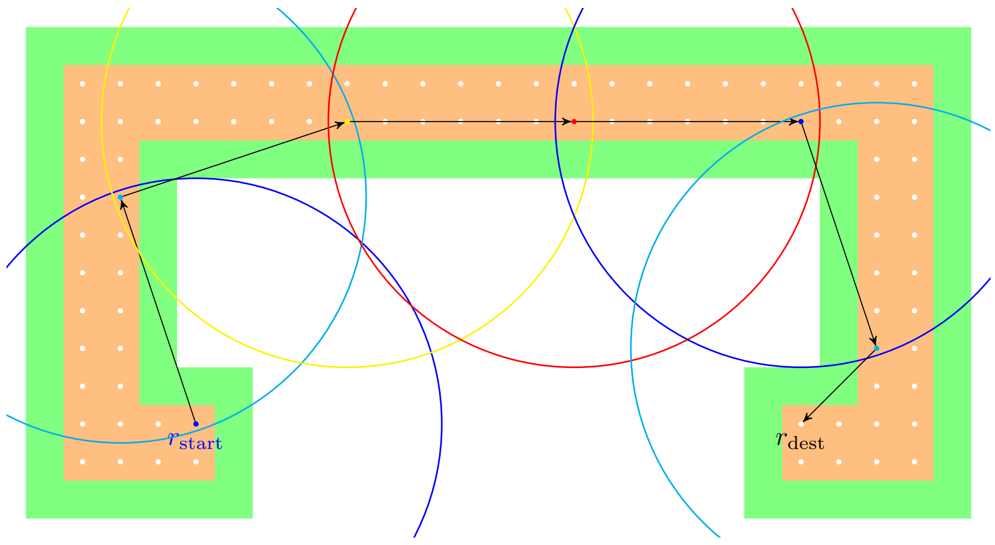

Figure 3: Robot movement v3 – Inteligent path without empty battery and slow movement.

4.1 Logistic Network as Graph

An interconnected logistics network of roboports can be viewed as a graph $G(N, E, \delta)$. Here, the roboports $r_1, \dots, r_n$ form the set of nodes N. The set of edges is defined as:

$$E = \{(r_i, r_j) \mid \forall r_i, r_j \in N : d(r_i, r_j) < \text{max\_range}(b_{\text{type}})\}, \tag{1}$$

where $d(r_i, r_j) = \|p(r_i) - p(r_j)\|_2$ determines the Euclidean distance between position p of the two roboports r_i and r_j, and $\text{max\_range}(b_{\text{type}})$ is the maximum range from the robot type currently considered. Separate graphs for robot types with different ranges may be needed.

The edge weight is mainly based on the actual distance between roboports, but a constant c is added to keep the number of intermediate stops and loading operations to a minimum:

$$\delta(r_i, r_j) = d(r_i, r_j) + c. \tag{2}$$

With $c = 0$ the undesired behavior could arise that more intermediate stops are made than necessary (cf. fig. 4). Therefore a $c > 0$ should be set, so the route in fig. 4a will be selected. Figure 5 illustrates that, in general, it is not always best to select the reachable roboport closest to the target.

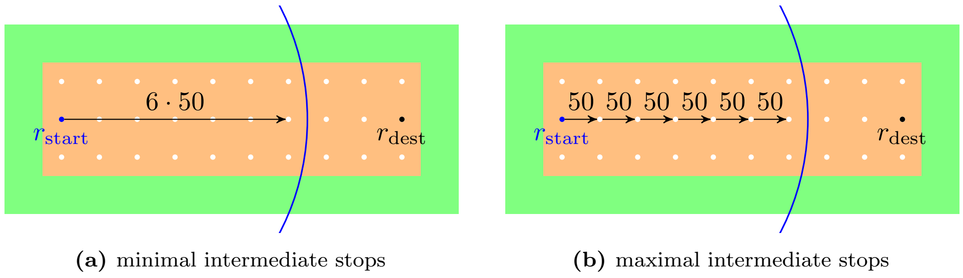

Figure 4: If only the roboport distance d would be considered, both routes would be equally good on the way to the destination r_{dest}. With a the roboport closest to the destination from r_{start} is approached directly, while with b 5 additional intermediate stops are made.

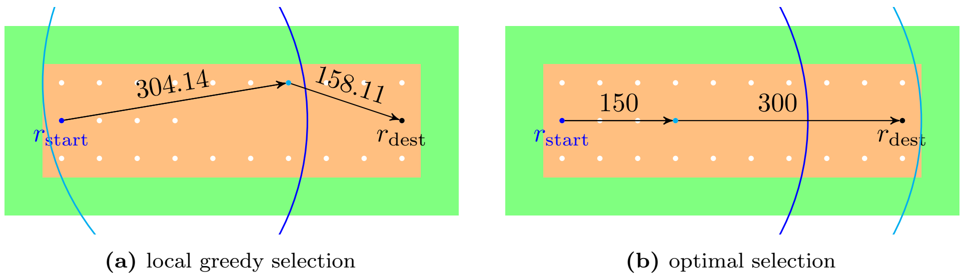

Figure 5: Two paths to the destination. For a, greedy selects the roboport closest to the destination, resulting in a total length of 462.25. For b, the total length of the path is taken into account and therefore the first intermediate stop is made early. This results in a smaller total length of 450.

4.2 Routing in the Graph

To find the shortest path within a graph, Dijkstra's algorithm can be used, for example. Using a Fibonacci heap, the runtime $\mathcal{O}(|V| \log(|V|) + |E|)$ can be achieved [Cor09, p. 662]. Importantly, this routing algorithm needs to be run globally only once per graph (on each node). An update only occurs when a roboport is added or removed (see P3). The algorithm can take place in a concurrent thread and thus should not negatively affect the UPS (solves P1). Since the graph for a logistic network and a robot type (logistic, construction and maybe more with mods) only has to be considered once and not somewhat per robot, the required memory is kept within limits.

As a result of the routing, a forwarding table is available at each node or roboport, which is used in the following. It would also be conceivable to use other routing methods than Dijkstra. In case of a very large number of Roboports, it could be useful to group them in the forwarding tables based on their position. Thus there would be an assignment of target area to next Roboport.

When selecting the next roboport that should also be used for charging, the procedure presented in FFF 374 [kOK23] under the section “Better robot charging heuristic” should be used.

4.3 Procedure of a Robot

Now consider the behavior of a single robot b that is to fly intelligently to a given target position g. The robot would *not* execute pathfinding algorithm (solves P1) and also *not* store the complete path to the destination (solves P2).

Instead, the robot must first determine the current next roboport. In most cases this is easy, since the robot is in this roboport if it has not just been manually placed in the air. This means that this information is usually already available in the form of a reference. Then the next intermediate target (Roboport at which loading is necessary) is looked up in the forwarding table on the way to the actual target. If the destination is already in range, it is approached directly. Only after charging at the intermediate destination, the next intermediate destination is looked up again in the forwarding table. In this way, the system also automatically reacts to changes in the network P3. The only exception would be if exactly the current destination roboport would be removed. Only in this case an alternative would have to be searched for, which however can be looked up in the meanwhile updated routing table of the predecessor node.

Since the routing tables were created in a separate thread as described above, only the time for looking in the “look up table” is needed here, which should be small. This procedure is roughly described in algorithm 2.

Algorithm 2: Robot rounting

```
Data: robot $b$, goal $g$
$b_{\text{dest}} = g$;
while $b_{pos} \neq b_{dest}$ do
    $r$ = next_roboport($b$);
    $b_{\text{intemediate\_dest}}$ = routing_table($r$, $g$);
    move_to($b_{\text{intemediate\_dest}}$);
end
```

4.4 Runtime Estimation for Different Scenarios

Now let’s go through the three scenarios from before again and compare them (s. section 1.1).

In the S1 scenario, only the first check would determine that the target is directly reachable and then fly to it directly. This is a calculation with position data, which is already available at the CPU for the movement calculation and therefore should be determined very quickly. Saved would be however the constant examination (or at least unique computation) whether the accumulator is empty! Thus this scenario should not take longer.

With S2 the next destination must be looked up once in the appropriate routing table for each intermediate stop. In the previous procedure, however, as soon as the battery is empty, a next Roboport must always be determined, which should be comparable in terms of effort.

If something changes in the logistics network and this does not affect the next roboport that is currently being addressed, this is automatically taken into account by parallel routing procedures. If exactly the current destination roboport is removed (S3), the next roboport must be determined or looked up once again. This is likewise comparable with the renewed search for the next Roboport, which takes place also with the past procedure.

5 Conclusion

Overall, the new routing approach seems promising. However, it should be tested with the implementation for different scenarios to be able to analyze the actual runtime behavior.

References

[Cor09] Thomas H. Cormen, ed. *Introduction to Algorithms.* 3rd ed. Cambridge, Mass: MIT Press, 2009. 1292 pp. ISBN: 978-0-262-03384-8 978-0-262-53305-8.

[kOK23] kovarex, Oxyd, and Klonan. *Friday Facts #374 - Smarter Robots*. Factorio. Sept. 1, 2023. URL: https://factorio.com/blog/post/fff-374 (visited on 09/02/2023).

[OTK19] Oxyd, TOGoS, and Klonan. *Friday Facts #317 - New Pathfinding Algorithm*. Factorio. Oct. 18, 2019. URL: https://factorio.com/blog/post/fff-317 (visited on 09/02/2023).