Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

More Shuttles, Less Cost: Energy Efficient Planning for Scalable High-Density Warehouse Environments

Christian Hütter
Department of Simulation and Graphics
Faculty of Computer Science
Otto-von-Guericke University Magdeburg, Germany

Abstract

We propose planning strategies for a shuttle-based warehousing system. Within this system, goods are not only transported by, but also stored on shuttles. This allows the simultaneous motion of all goods without preparation time. We exploit these properties to improve both space- and energy efficiency compared to conventional storage systems while maintaining fast access to all goods.

We present a framework to manage the shuttles in a very high density environment, allowing efficient in- and output, storage, and sequencing. The proposed framework is conflict-free, space- and energy efficient and its practical applicability is demonstrated in a simulative analysis based on a real-world problem. In our sample application holding 1950 shuttles we achieve storage densities of approximately 88% while maintaining efficient access to all goods.

Introduction & Overview

With the advent of shuttle-based systems like "Amazon Robotics" (formerly Kiva Systems), "MultiShuttle Move", "Autostore" and others, a new generation of material handling systems is emerging. However, the trend towards individual goods handling has not yet reached its turning point: we see the potential for these technologies to converge into a system where each good is not only transported, but also stored on an individual shuttle (patents for such systems have been granted already[1]). This allows all objects to move simultaneously and without preparation time.

To illustrate a possible deployment of such a system, let us consider the following sample application. At an airport, the baggage of passengers arriving early or changing aircrafts needs to be stored temporarily. This is done in an "early baggage store". Single pieces of baggage, each placed on a baggage tray, arrive at the store in random intervals. Upon their arrival, the baggage is put in a buffer, called a delay line, where it waits for a free spot on a connected conveyor belt. This conveyor, shown in fig. 1, is the actual storage space; it moves the baggage in circles. On it, the baggage passes an output junction with every rotation of the conveyor. Once requested, the junction is used to remove the baggage from storage. While baggage input occurs in random intervals,

[1] http://www.google.com/patents/EP2165406B1?cl=en

Figure 1: A conveyor is used to store baggage trays while providing random access. This scenario can be vastly improved with the system shown in fig. 2

the removal of baggage belonging to one flight is performed in blocks of approximately 20 pieces at a time once the flight is ready. Due to delayed flights, passengers changing flights last-minute etc., we do not know the removal order in advance. Instead, we require access to any individual piece of baggage within a limited amount of time. For this reason, the baggage is moved all the time: not to store it, but to solve the sequencing task, and separating both aspects is not easily achieved with conventional material handling systems. The system must hence make a trade-off: decreasing the baggage's speed reduces energy consumption, but increases access times. This also means that a larger store needs to increase its speed to maintain access time, a limiting factor for scalability. To combine storage (with no need for motion) and sequencing task (with need for little motion), we have to create a system which performs a lot of motion when utilizing this conventional system.

We employ a shuttle-based system to mitigate these effects, moving baggage only when needed, improving energy efficiency and scalability, while simultaneously reducing space requirement. The system we consider is shown in fig. 2. Instead of using a linear conveyor, the shuttles are capable of moving in two dimensions, taking a single step in x or y direction at a time. This allows each piece of baggage to stop moving for the major part of its storage duration, as other shuttles can move around it. We assume that each piece of baggage is arriving on its own shuttle and removed from the system together with it (emulating the behavior of

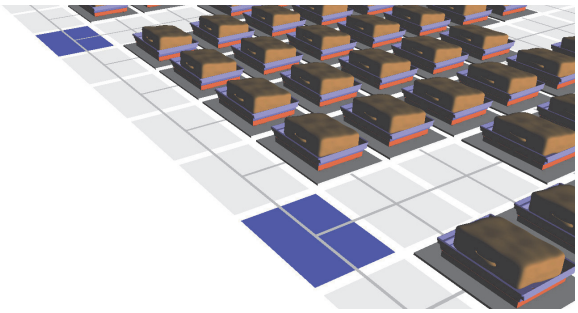


Figure 2: We propose a system where each baggage tray is placed on a shuttle. Shuttles can movie between cells in the plane, but remain stationary whenever possible.

the baggage trays in today's system).

The properties of the system are as follows. We consider rectangular shuttles of equal size placed in a grid. A shuttle can move from its current grid location to one in its 4-neighborhood. Shuttles are allowed to move simultaneously, provided their respective destinations are free. Also, several shuttles standing edge to edge in a row or column, travelling in the same direction, may move at once, similar to objects moving on a conveyor. That is, moving a row of shuttles one cell to the right can be done in the same amount of time that is required to move a single shuttle one cell to the right. The restrictions of geometry and position are imposed to avoid exponential running time: deciding if we can arrange rectangular objects (of arbitrary size) inside a rectangular workspace (without restricting them to grid locations) in a given configuration is PSPACE-hard (Hopcroft, Schwartz, and Sharir 1984).

Under these assumptions we solve the task of putting a shuttle from an input cell into storage, thus clearing the input cell for a subsequent input operation. As with all operations we consider, this is solved for high-density situations where no free path exists between input- and storage cell. We then prepare a set of shuttles for removal from storage, putting one of the prepared shuttles on an output cell for immediate removal and the remainder "close to it" for fast subsequent output operations. An important operation we use to solve both tasks is the computation of an energy optimal motion sequence clearing a given set of cells w.r.t. a unit cost model.

The underlying conditions for a system acting as early baggage store are sufficient storage capacity, in- and output speed and reliability. The optimization criteria are the minimization of required space and energy as well as the number of interconnection (i.e., in- and output) cells.

To achieve these goals, a predictable and efficient planner is required, capable of handling thousands of shuttles whose motions are interdependent due to the high storage density. By omitting some of the inputs commonly provided for motion planning tasks, foremost the destination of each individual shuttle, the planning task comprises not only motion-, but as an additional challenge also destination planning. Finally, to achieve good results it does not suffice to handle each single task as an isolated planning problem. Instead, we must also consider parallel task execution and ensuing tasks, which might draw a benefit or suffer a penalty from the choices made in the current planning step.

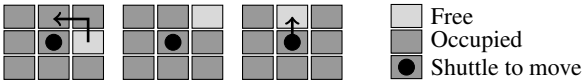
Using the techniques we discuss in this paper, the proposed shuttle-based system fulfills the underlying conditions and proves a high potential regarding the optimization goals: instead of moving each piece of baggage $24\,000\,\mathrm{m}$, as is currently done in the conventional system, we move it–on average–only approximately $20\,\mathrm{m}$, reducing both energy consumption and maintenance costs. At the same time, the space requirement for the system is reduced from approximately $5150\,\mathrm{m}^2$ to $2250\,\mathrm{m}^2$.

Related Work

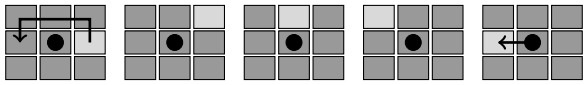
To manage a warehouse operated by a fleet of shuttles with only little maneuvering space, a cooperative planning approach is required: without it, a shuttle with no direct road-map access is "stuck" at its position as it cannot reach locations outside its currently accessible configuration space on its own. Such cooperation is achieved by coupled, centralized planners, which regard the fleet as a single, composite robot with many degrees of freedom (Parsons and Canny 1990). The exponential running time of these coupled, centralized planners, however, makes them unsuitable for the large fleets we consider.

By sacrificing the optimality offered by these planners, complete, polynomial-time algorithms become available for certain environments. To move a shuttle with no road-map access to a specified (output-) position, an evasive motion of the shuttles in its way suffices. Such motions have been studied e.g. by (de Wilde, ter Mors, and Witteveen 2013), who improved an earlier approach by (Luna and Bekris 2011). The idea is to "push" and "swap" or "rotate" shuttles from their current location to their destination, requiring only two free cells. Their focus is on the so-called pebble motion problem: in a graph where each vertex may be occupied by at most one labeled pebble, the task is to create a new pebble configuration by consecutively moving pebbles to adjacent (unoccupied) vertices. They present an efficient and complete algorithm solving the pebble motion problem for instances with at least two empty vertices. The motions described in their work involves the shuttle which is to be moved and its neighbors; it is therefore not always possible to use it to clear a road-map in advance This makes its execution relatively slow in our scenario, as the shuttle we want to move needs to wait for the next vertex to become cleared at each step.

Another approach to move any individual shuttle to a given (exit-) grid location was proposed by (Gue and Kim 2007). The motion sequence executed to move a shuttle one cell forward along a given, non-empty path, consists of 3 to 5 steps, called an "escort moves", depicted in fig. 3. While this straight-forward solution requires only one free cell, the ramifications are similar: While the clearing motions are executed, the requested shuttle has to wait, which has a huge impact on speed. In addition, 3 to 5 steps are performed to move a single step forward, leading to a high energy con-



(a) An escort move for moving a shuttle around a corner.



(b) An escort move for moving a shuttle along a straight path.

Figure 3: Escort Move, as depicted in (Gue and Kim 2007)

sumption[2].

A planner allowing the parallel execution of clearing motions was proposed by (Peasgood, Clark, and McPhee 2008). However, it requires the computation of a spanning tree and can handle at most as many shuttles as the spanning tree has leaves. That leads to the new problem of finding a maximum-leaf spanning tree, which in itself is NP-complete (Garey and Johnson 1979). Also, the number of inner nodes–being the minimal amount of free cells required–is quite large and the energy consumption suboptimal.

When we consider the shuttles between current position and destination as obstacles, our task becomes a "planning in the presence of movable obstacles"-problem. While extensive research has been conducted in this area, a key characteristic of the problem as stated in the literature is that the obstacles themselves are passive. That is, it does not suffice to compute their trajectory. Instead, the planning robot must manipulate the environment itself (Ben-Shahar and Rivlin 1998; Stilman et al. 2007; Lindzey et al. 2014). Not all means of manipulating can be transferred to our application–e.g., grasping, sweeping and toppling as proposed by (Dogar and Srinivasa 2011)–or are unnecessarily complex (e.g., pushing). In conclusion, these planners do not adequately exploit the benefits that the self-movable obstacles in our scenario provide.

Planning Framework

We start by defining the general framework and proceed by listing the planning problems we solve. Let $G = \{(x, y) \subset \mathbb{N}^2 : 0 < x \leq W, 0 < y \leq H\}$ be a rectangular grid with width $W > 1$ and height $H > 1$ of cells with integer coordinates, S a finite set of shuttles and $P : S \to G$ an injective position assignment of each shuttle to a cell. We define our state space $\mathcal{P}$ as the set of all possible position assignments. For a set $T = \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$ of unit steps let $U : \mathcal{P} \to 2^{S \times T}$ via $U(P) = \{(s, t) \in S \times T : s \text{ can move to } P(s) + t\}$ be the set of available transitions; we say s can move to $P(s) + t$ iff $P(s) + t \in G$ and $P^{-1}(P(s) + t) = \emptyset$.

We produce a state $P' \in \mathcal{P}$ from an adjacent state $P \in \mathcal{P}$ by moving a single shuttle to a free adjacent cell using a state transition function:

$$P'(x) = p(P, (s, t))(x) = \begin{cases} P(x) & x \neq s \\ P(x) + t & x = s \end{cases}$$

for a (shuttle,step)-pair $(s, t) \in U(P)$. Under certain conditions we perform multiple transitions simultaneously: Let $(s_1, t_1), \ldots, (s_k, t_k)$ be a sequence of transitions and $P_0, \ldots, P_k$ a corresponding sequence of position assignments. We execute all transitions (s_j, t_j) at the same time as transition $(s_i, t_i) \forall i$ and $j > i$ if three conditions are met: (a) no transition involving s_j must exists between transitions i and j, (b) the destination cell $d := P_j(s_j)$ is not used between transitions i and j by another shuttle, i.e. $\forall l : i \leq l < j : P_l^{-1}(d) = \emptyset$, and (c) either d is already free before step i, or d is the position of s_i before step i and s_i and s_j move in the same direction: $P_{i-1}^{-1}(d) = \emptyset$ or $P_{i-1}^{-1}(d) = s_i \wedge t_i = t_j$. That is, we allow simultaneous motion of multiple edge-to-edge shuttles in the same direction.

Furthermore, let $I \subseteq G$ be a set of input cells and $O \subseteq G$ a set of output cells; shuttles may be added or removed to and from the system at these cells only, respectively. For a configuration $P \in \mathcal{P}$, let $E(P) = \{g \in G : P^{-1}(g) = \emptyset\}$ be the set of unoccupied cells. Last but not least let $P_0 \in \mathcal{P}$ be the initial configuration and $((s_1, t_1), \ldots, (s_k, t_k)) \in U(P_0) \times \ldots \times U(P_{k-1})$ a transition sequence or *plan*. When we say a plan produces some postcondition, e.g., clears location (x, y), we imply that applying the transition function to the plan's transitions, starting from state P at the time of planning, has the described effect. As cost function we use the length of a plan in transitions, which corresponds to the energy consumption when using a unit cost model.

We solve the following tasks:

Storage Task An empty input location is required to add a new shuttle to the system. For an input location $(x, y) \in I$ which is nonempty, i.e. $P_0^{-1}((x, y)) \neq \emptyset$, we find a plan that clears (x, y), such that $P_k^{-1}((x, y)) = \emptyset$. This operation requires one free cell, which will be used as destination.

Clearing Task For a given set $C \subseteq G$ of cells with $|C| \leq |G| - |S|$, we find a plan that clears C: $P_k^{-1}(C) = \emptyset$. Note: Problem 1 is an instance of problem 2 with $C = \{(x, y)\}$ for an input location $(x, y) \in I$.

Output Preparation For a given subset of Shuttles $D \in S$ and output $(x, y) \in O$ we find a plan that moves a nonempty subset of D to a set of connected cells in G that includes o.

Input Operation

To observe the underlying condition of sufficient input speed, we must clear the input cell quickly to allow subsequent inputs. In addition, to minimize energy consumption, we want to choose a storage cell with a minimal combined in- and output cost. Unfortunately, no generic strategy for choosing this location fits all applications as most strategies change their behavior depending on the precise input/output sequence. This is especially true for scenarios

[2]For the sake of simplicity we use a unit cost model for energy consumption: travelling to an adjacent cell costs one energy unit, disregarding acceleration etc.

without a road-map connecting all storage cells. In such scenarios, evasive motions change the locations of stored shuttles, making different cells available for subsequent inputs. The in-/output sequence is unknown in our example, as it is in most real-world applications.

We therefore propose a greedy approach working as follows. For a storage task with given inbound location $(x_i, y_i) \in I$, we find the closest outbound location $(x_o, y_o) \in O$. Note that all distances are w.r.t. the grid G; for a unit cost model, where a transition (s, t) has cost 1, this is equal to the Manhattan distance of the respective grid points. We then label each cell $(x, y) \in G$ with its combined distance from in- and outbound location $\|(x, y)-(x_i, y_i)\|_1+\|(x, y)-(x_o, y_o)\|_1$ as shown in fig. 4a and choose a free cell d among those with smallest label. We find a shortest path from i to d, disregarding all movable obstacles, i.e., stored shuttles. Along this path, we perform what we call a "push move": each shuttle on the path moves towards d (as if "pushed" by the shuttle on i), thereby clearing the inbound position and using d as a new storage cell. The shuttle is now stored as shown in fig. 4b.

We define a push move plan $R = ((s_1, t_i), \ldots, (s_k, t_k))$ for a path $M = (g_1, \ldots, g_n)$ with $P_0^{-1}(g_1) \neq \emptyset$ and $P_0^{-1}(g_n) = \emptyset$ as follows:

- R clears the source cell and occupies the destination cell: $P_k^{-1}(g_1) = \emptyset \wedge P_k^{-1}(g_n) \neq \emptyset$
- No other cell changes its occupancy status: $\forall g \in G \setminus \{g_1, g_n\} : P_0^{-1}(g) = \emptyset \Leftrightarrow P_k^{-1}(g) = \emptyset$
- Shuttles not on the path do not change their positions: $\forall g \in G \setminus M : P_0^{-1}(g) = P_k^{-1}(g)$

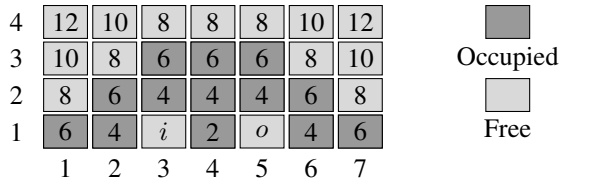
Algorithm 1 constructs a push move. Going back from the destination cell g_n, we set the dst-pointer to the last free cell (which is g_n in the beginning) and the src-pointer to the last non-free cell on path M (lines 4-6). We then add transitions to move the shuttle found on src to dst (lines 7-11). We repeat this procedure with dst set to the cell cleared by these transitions, i.e., src (line 12), until g_1 is cleared (line 3).

Input: Path $M = (g_1, \ldots, g_n)$ as a connected grid position sequence with $P(g_1) \neq \emptyset$ and $P(g_n) = \emptyset$
Output: Push move plan R

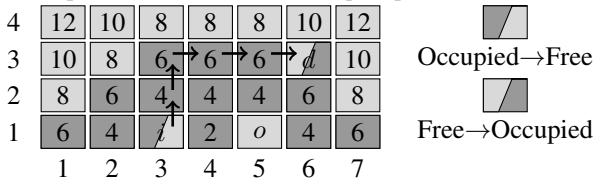
```
1:  R ← ()
2:  dst ← src ← g_n
3:  while src ≠ g_1 do
4:      while P_0^{-1}(src) = ∅ do
5:          src ← predecessor of src on M
6:      end while
7:      t ← src
8:      while t ≠ dst do
9:          R ← R ∘ (P_0^{-1}(src), successor of t on M − t)
10:         t ← successor of t on M
11:     end while
12:     dst ← src
13: end while
```

Algorithm 1: Push Move

The cost for the push move plan is equal to the path length



(a) Each cell is labelled with the combined distance from i and o. The occupation after 14 in- and no output operations is shown.



(b) The next input operation moves all shuttles in row 3 to the right (simultaneously), then the ones in column 3 up. This clears i and occupies d.

Figure 4: To input a shuttle from input cell i using storage cell d we "push" each shuttle on an i, d-path towards d, thus clearing i.

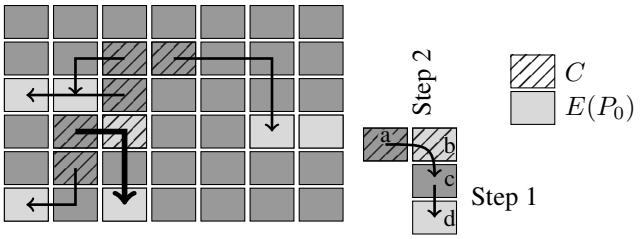
in edges, as each edge on the path corresponds to one transition of the plan. That is, the cost of moving a single shuttle from g_1 to g_n is equal to the cost of performing a push move from g_1 to g_n. The latter alters the positions of all shuttles on the path, not only of the one on g_1, which may or may not be a disadvantage depending on the scenario. However, as a huge benefit, it does not require the path to be free, and the effect regarding occupancy is the same: g_1 becomes free, g_n becomes occupied, and occupancy of all cells in between remains unchanged.

Finally, let us look at the time required to clear g_1. In a rectangular grid where we are allowed to move any shuttle, we can find a shortest path from any position g_1 to any other position g_n consisting of one horizontal and one vertical segment. All shuttles on either segment can move simultaneously–in fig. 4b, we first move all shuttles in row 3 to the right simultaneously, and then all shuttles in column 3 up. Hence, the input cell is cleared after the time required to perform only two transitions.

If shuttles are removed starting with the ones closest to the output cell–thus avoiding the need for evasive motions–our greedy strategy is optimal: the total combined input/output cost is equal to the sum of all labels used for storage, which is an obvious lower bound for the given storage task and layout combination.

(Path-) Clearing Algorithm

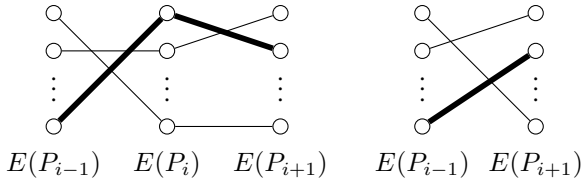
The previously defined "push move" operation clears a single cell. In this section, we use it to clear a set of cells $C \subseteq G$. We define a C-clearing motion as a plan which clears C, that is $P_k^{-1}(C) = \emptyset$. This obviously requires $|C| \leq |E(P)|$. In case of $|C| < |E(P)|$ the position of some free cells remains unchanged. To specify exactly which free cells shall remain unchanged, we may include a subset of $E(P)$ in C, making sure these included cells remain free



(a) To clear the cells is C, we perform a push move for a shortest path between each pair in a C-$E(P_0)$-matching.

(b) By moving the shuttle from a to c, the push move preserves free cell b.

Figure 5: We clear the hatched cells C with push moves from each cell in C to one in $E(P_0)$.



(a) The matchings of two free cell sets produced by two consecutive transitions in a plan...

(b) ...are merged for cost equal to the sum of the individual matchings.

Figure 6: Each transition of an optimal C-clearing plan induces a matching between $E(P_i)$ and $E(P_{i+1})$. All of these matchings can be combined.

after the operation.

The algorithm works as follows. We first compute the cost minimal, inclusion maximal matching M between C and $E(P)$. As cost for matching $c \in C$ to $e \in E(P)$ we choose the length of a shortest c, e-path (or 0 for $c = e$). For each pair $\{c_m, e_m\} \in M$ of the matching we compute a shortest c_m, e_m-path. Along each such path we perform a push move as described in the previous section. All cells c_m from the matching are thus cleared. All cells in C are part of the inclusion maximal matching as G is connected and $|C| \leq |E(P)|$, hence C is cleared as claimed. The cost of the resulting plan, comprising all push moves, is equal to the cost of the matching as each c_m, e_m-push move has cost of its corresponding c_m, e_m-path's length (cf. Algorithm 1). Figure 5 illustrates a clearing motion.

We claim that this strategy for clearing C is optimal w.r.t. energy consumption when using a unit cost model. Let $X = (s_1, t_1), \ldots, (s_k, t_k)$ be a shortest, i.e., energy optimal C-clearing plan. Each transition (s_i, t_i) from X moves s_i from $P_{i-1}(s_i)$ to $P_i(s_i)$. It thereby changes the empty cell set from $E(P_{i-1})$ to $E(P_i) = (E(P_{i-1}) \cup P_{i-1}(s_i)) \setminus P_i(s_i)$. $E(P_{i-1})$ and $E(P_i)$ only differ in this one element, hence a perfect cost 1 matching exists between $E(P_{i-1})$ and $E(P_i)$. We construct this matching by assigning $P_{i-1}(s_i)$ to $P_i(s_i)$ for cost 1 (as these cells are adjacent) and every other element to "itself" for cost 0.

Using this technique, we get a series of k matchings–from $E(P_0)$ to $E(P_1)$, from $E(P_1)$ to $E(P_2)$ and so on, one for each transition in the optimal transition sequence X. In this series of matchings the destination $E(P_1)$ of the first matching is the same as the source of the second matching. Therefore, they can be merged, as shown in fig. 6. By combining all k matchings in the series we get a cost $k = |X|$ matching that matches $E(P_0)$ with $E(P_k)$.

We chose X to be a C-clearing motion, hence $C \subseteq E(P_k)$, and the k combined matchings match a superset of C to the free cells for cost k. Therefore, a matching of cost (at most) k between C and the free cells $E(P_0)$ exists. This proves the claim that our approach is (at most) as expensive as the optimal C-clearing plan X. We already argued that our approach clears C for the cost of this matching, thus completing our proof.

In some scenarios we want to "undo" the shuttle displacement created by clearing C. For example, if we use the algorithm to clear a path in a non-chaotic store, we alter some shuttle locations. These locations must ultimately be restored, as the store is non-chaotic. This is achieved by reversing the direction of each transition of the C-clearing plan and performing them in reverse order. For the C-clearing plan $A = \{(s_1, t_1), \ldots (s_k, t_k)\}$ we construct $A^{-1} := \{(s_k, -1 \cdot t_k), \ldots, (s_1, -1 \cdot t_1)\}$. This method only works if no cell which is part of the C-clearing motion changes its assigned shuttle between executing A and A^{-1}. In many instances, however, we want to use a road created by clearing C to move some shuttle s to a destination d, and when reversing the operation, we want s to stay at d. In order to do so, we must ensure that neither s nor d is part of the C-clearing motion. This is achieved as follows. First, if d isn't free already, we clear d. Then, we remove $P(s)$ and d from the cell grid G, after which we compute the clearing motion as described above. With $P(s)$ and d removed, no path used in a push move contains these cells, hence we ensure that they do not take part in the clearing motion (or its reverse) at all. Finally, the cells can be re-added and s moved to d, after which A^{-1} is applied. As A^{-1} does not contain $P(s)$ or d, it is unimportant whether or not they are removed while executing the plan. When removing several cells $D = \{d_0, \ldots, d_n\}$ from G before computing a clearing motion, we must take care that each connected component of $G \setminus D$ has at least as many free cells as it has cells in C.

In the next section, we use the technique of removing cells from the grid before computing clearing motions to "fix" the position of some shuttles. In practice, the same can be done to prevent interference with input or removal operations, ensuring that shuttles being processed do not move unintentionally.

Output Preparation

We now use the clearing algorithm to prepare the removal of shuttles from storage. To this end, we create an output buffer, which places one shuttle at output $o \in O$ and (possibly) additional shuttles we want to remove close to it. Then, whenever the shuttle on o is removed from the system, we place a new shuttles from the buffer on o.

An o, D output buffer for output cell $o \in O$ and a set of

requested shuttles $D \subseteq S$ is a connected set of cells $B_o \subseteq G$ with $o \in B_o$ in which each cell contains a requested shuttle, i.e. $\forall (x, y) \in B_o : P^{-1}((x, y)) \in D$. Note that at least one, but not necessarily all shuttles from D must be part of the buffer. An o, D output preparation task is a plan that creates such a buffer. When the shuttle on o is removed from the system (i.e., it is removed from S, D, and physically), o becomes free, shortly violating the output buffer condition that all output buffer cells must contain a shuttle from D. To restore the condition, we compute a B_o-spanning shortest path tree rooted in o. We then choose an (arbitrary) leaf l in this tree and perform a push move on the l, o-path in the tree. The push move clears leaf l and we remove it from the output buffer. At the same time a new shuttle from D is placed on o. Once the last leaf is removed, either D is empty and the output is complete, or a new output preparation task for o and the remainder of D is performed, until all shuttles in D are removed from storage.

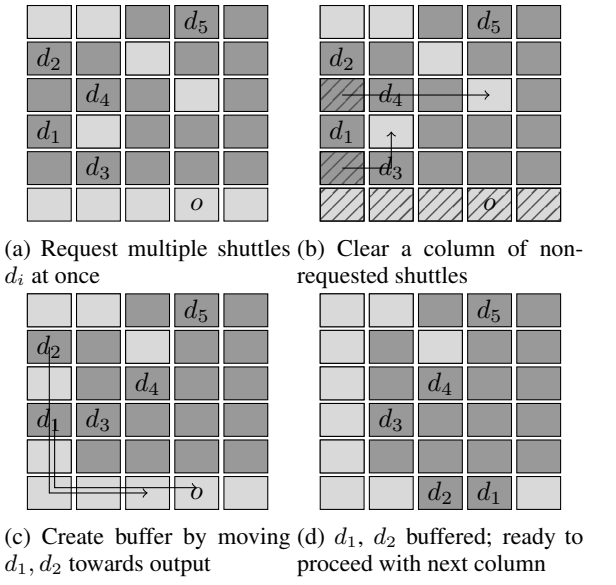
We compute an o, D output preparation task as follows. First, we find a shortest $P(d), o$-path for some shuttle $d \in D$. Let M be the set of cells on that $P(d), o$-path. We partition $M = M_D \cup M_C$ into cells occupied by a shuttle in D, that is $M_D = \{g \in M : P^{-1}(g) \in D\}$ and all other cells, i.e. $M_C = M \setminus M_D$. We then compute a M_C-clearing motion in $G \setminus M_D$. Removing M_D from G before computing the clearing motion ensures that no shuttle from D which is located on the path changes its position due to the clearing motion. By executing the clearing motion, we clear the path of all shuttles not belonging to D. Thus, the remaining shuttles on the path all belong to D. We now move these remaining shuttles towards o until they are placed on a connected subset of M which includes o and contains no free cell, thus forming an output buffer.

In the experimental section we use a layout where all output cells are placed at the bottom row. This row is not used for storage purposes. That is, when we perform an input task or clearing operation, we do not consider the bottom row as possible target locations. For an output location $(x_o, 1) \in O$ we choose a column with $x \leq x_o$ containing at least one shuttle from D. We use this column as the vertical segment together with the bottom row as the horizontal segment of our $P(d), o$-path for the output buffer creation. We then do the same for a column with $x > x_o$, providing a second branch in the shortest path tree of the output buffer. In this manner we execute output preparation tasks for a column to the left and right of the output in alternation before the respective other branch is empty, allowing continuous output at most times. We illustrate this column-based strategy to create an output buffer in fig. 7.

We have now discussed methods for solving the storage-, clearing-, and output preparation tasks stated as our problem definitions. In the last part before the experimental section we take a look at layout considerations, the placement of in- and output cells and scalability.

In- and Outbound Locations

The placement of in- and output cells has a major influence on all key aspects of our system. As this placement is largely independent of the overall plant design, it stands to reason



(a) Request multiple shuttles d_i at once

(b) Clear a column of non-requested shuttles

(c) Create buffer by moving d_1, d_2 towards output

(d) d_1, d_2 buffered; ready to proceed with next column

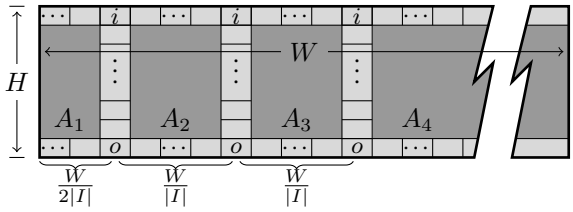
Figure 7: Column-based output buffer creation. We use the bottom row exclusively as output buffer.

that we can influence it within certain bounds. Assuming this flexibility, what are good positions for in- and output cells? We first should consider some restrictions: the in- and output cells must be placed at the border of the grid, placing them in the middle is usually not an option. Also, using the same cell for both in- and output is not possible in many situations, as different subsequent systems are connected. Other than that, we want to minimize energy consumption and the number of required interconnection points (i.e., minimize $|I| + |O|$) while maintaining the required performance.

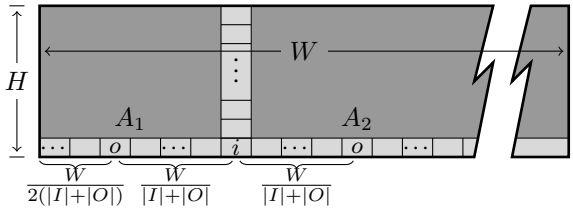
The typical layout provides input at one side and output at the other side of the storage facility. This layout is often due to the benefit created by one-way roads in conventional storage facilities. In our case, one-way roads create little to no benefit, as we perform input operations via push moves, not requiring roads at all.

If our storage is not fully utilized, we do not occupy all storage positions, but rather choose those with small labels. In a typical storage, the design capacity exceeds the average utilization by far. In our example, the design capacity is 1950 pieces of baggage, but at most 595 pieces are stored at once on the average day (with significantly lower numbers in the morning and evening). Hence, to minimize energy consumption, we want to choose I and O in a way that creates many "cheap" locations, i.e., cells with low label. This is achieved by placing in- and outbound locations at the same side of the grid. In doing so, we reduce the distance of in- and output locations, i.e. the minimal label value, when compared to a layout with I and O placed at opposing sides of the grid.

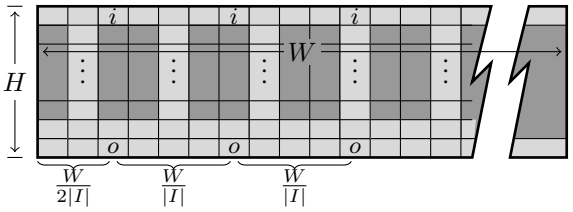
If the push-move paths of multiple input operations overlap, they become interdependent, and not all transitions of the push moves can be executed simultaneously. This re-



(a) In- and output cells placed at opposite sides, allowing one-way roads optimizing speed.



(b) In- and output cells placed at the same side, minimizing their distance to optimize energy consumption in average load situations.



(c) Naïve layout used as point of reference and comparison to conventional shuttle-based systems.

Figure 8: The three layouts used in the simulation runs.

duces overall input speed and impairs scalability. The same problem occurs in the output preparation task when the push-move paths of the clearing-motions, used for output buffer creation, overlap. To eliminate both in- and output dependencies, we partition our grid into parts containing one in- and one output each, basically dividing G into individual stores, which we call *areas*. Each area provides enough free cells to create its own output buffer, eliminating the need to perform push moves between different areas. We still allow input operations to use push move paths spanning multiple areas if the area assigned to the input cell is already full. However, for input operations distributed uniformly among all input cells, this is an uncommon event.

In the remainder of the paper we demonstrate the practical applicability of our approach in a simulation using real-world data.

Simulation Model

We use early baggage store data of a major European international airport to evaluate our system. In a nutshell, the baggage store works as follows. Pieces of baggage arrive at random intervals at given inbound locations. When a piece of baggage assigned to input cell $(x, y) \in I$ arrives, two possibilities arise: either (x, y) is free, then we add a new shuttle s to S with $P(s) = (x, y)$, or (x, y) is occupied by another shuttle, then we add s to an input queue associated with (x, y). In the latter case, s will be added to S as soon as (x, y) is free and no shuttles precede it in the given input queue. Once the shuttle is added to S, we wait three seconds to simulate a loading operation, after which we clear (x, y) by executing a storage task.

Each piece of arriving baggage has a flight number. We use the flight number to partition S into request sets: $S = \{F_1 \cup \ldots \cup F_n\}$. At some point in time (the "begin baggage transport"-time), a buffer outside the early baggage store is assigned to the flight. In this buffer, the baggage for the flight is collected before being removed from the baggage handling system. However, in the general case the capacity of this buffer is less than the capacity of the airplane. We therefore cannot remove all stored baggage associated with the flight at once. Instead, the baggage for one flight is requested in several blocks over 30 minutes to 3 hours, mainly depending on the flight size and the portion of baggage placed in the early baggage store. We should note that all output cells are connected to the same conveyor; we therefore can choose the output location freely. The layouts we analyze are divided into storage areas allowing independent in- and output, as described in the previous section, labelled A_i in figs. 8a and 8b. When k shuttles for flight F_i are requested for output, we proceed as follows. For each output $o \in O$ we select a column $C_o := (x_p, y) : 0 < y \leq H$ belonging to the respective output's storage area that contains the most shuttles in F_i. We then perform for each output $o \in O$ an $o, P^{-1}(C_o) \cap F_i$-output preparation task. If a storage area for an output does not contain shuttles belonging to F_i or if k is exceeded, we skip the output or end the procedure, respectively, and only prepare a subset of shuttles from the last column for output so as not to exceed k. The shuttles placed on the output cells by the output preparation task are removed three seconds after they finished their last transition to simulate an unloading operation. After their removal, we place the next shuttle from the buffer on the output cell as described in the "Output Preparation" section. If less than k shuttles were prepared, we execute the next output preparation task before the output buffer is empty, so as to minimize the idle time of output cells.

Careful analysis of current real-world data shows that the time between two flights (i.e., between two "begin baggage transport"-signals) can be modelled as a non-stationary Poisson process. Flight size follows a time-dependent beta-binomial distribution. Baggage input times, too, follow a non-stationary Poisson process.

To validate the layout w.r.t. storage capacity and performance we use the data of maximum-load days (w.r.t. all days of the year 2014) as simulation input. Once sufficient performance is verified, data for average-load days is used to determine expected energy consumption. This is important as larger storage facilities usually imply higher energy rates to operate, a property we tried to mitigate with the layout shown in fig. 8b. The design capacity is 1950 shuttles/pieces

of baggage. On the average day, 4941 pieces of baggage are removed from storage. The amount of added baggage is almost equal, but due to overnight baggage not precisely the same. Baggage that entered the store before 00:00 is present in the simulation, but its motions before 00:00 are not counted. Similarly, baggage that is left in the store after 24:00 occurs, but is not included when computing averages. The used capacity on the average load day peaks at 595. On the maximum load day, 10040 pieces of baggage are removed from storage, and the used capacity is 1939.

Results

We performed simulation runs for three layouts, depicted in fig. 8, containing a same-side and two opposite-side I/O placement strategies. We implemented a naïve storage strategy depicted in fig. 8c as a point of reference. It stores the shuttles in columns with a road-map connecting all storage locations. Shuttles may not travel to the output cell as long as it is occupied and we cannot keep shuttles waiting for their removal on the road-map, as this would block access to other shuttles and free storage locations. We therefore add a buffer area near the output cells to achieve sufficient output speed in this layout. We would use a similar system for a "conventional" shuttle-, AGV or forklift-operated warehouse, as clearing a path through goods without own actuators would be too time- and energy consuming. Layouts (a) reserves two, layout (b) only one row which are not used for storage (except as output buffer). Similarly, all columns containing an input cell are only used to move shuttles, not to store them. Each storage area A_i in fig. 8 is associated with one or two outputs in layout (b) or (a) respectively and reserves, for a H cell heigh layout, $H-1$ cells to perform output preparation tasks. This way, all storage areas can perform independent, i.e. parallel, output preparations. The empty row connecting output locations greatly improves output speed, as we do not have to clear it in every output preparation task. The free columns at the input cells increase input speed by reducing the dependencies between output buffer creation and storage tasks.

We provide a comparison to the naïve approach (and, to a certain extent, the existing, not shuttle based system) as evaluating other, shuttle-based systems as reference proved problematic. The STRIPS system, as an example for a logic based, coupled & centralized planner, has exponential running time if used with our transition operator: to apply it, two preconditions must be met, namely the source cell must contain a shuttle while the destination cell must be free. Similarly, two postconditions hold after the transition, stating the reverse. (Bylander 1991) showed that the STRIPS problem with two pre- and two postconditions is NP-hard, making it unsuitable for the number of shuttles we have to handle. The polynomial-time planners discussed in the Related Work section, on the other hand, lack sufficient evidence that they meet the performance requirements on the logistics level. For example, the approach by (Gue and Kim 2007) requires 3 to 5 sequential transitions to move a shuttle by a single cell towards its goal. If a shuttle has an average distance of 10 cells to o and we assume 4 consecutive steps on average to move the shuttle one step forward, using this approach to move the shuttle to o takes the time of 40 transitions. Further assuming an average cell distance of $1\,\mathrm{m}$ and acceleration of $1\,\mathrm{m/s^2}$ we get a total transition time of $80\,\mathrm{s}$. For a $3\,\mathrm{s}$ shuttle removal time, we would need to move 27 shuttles simultaneously towards o (per storage area) to achieve near-optimal utilization of the outbound location. Which leads to the new problem of coordinating these motions in a conflict-free way while avoiding further delays.

Layout	Load	$\lvert I\rvert$	$\lvert O\rvert$	IQ	Steps	% used
1:10 (a)	max	4	4	4	31.6	82.7%
	avg	4	4	3	23.5	82.7%
(b)	max	4	5	52	31.7	87.6%
	avg	4	5	3	19.7	87.6%
(c)	max	5	5	35	33.6	53.1%
	avg	5	5	2	28.9	53.1%
1:6 (a)	max	5	5	5	35.8	81.5%
	avg	5	5	2	27.9	81.5%
(b)	max	4	5	91	30.3	86.9%
	avg	4	5	3	17.6	86.9%
(c)	max	5	5	18	36.8	57.5%
	avg	5	5	2	32.6	57.5%

Table 1: Simulation results.

All layouts use the same cell dimensions of $0.8\,\mathrm{m}$ by $1.2\,\mathrm{m}$, or approximately $1\,\mathrm{m^2}$ per cell. We assume that shuttles have a maximum speed of $2\,\mathrm{m/s}$ and a maximum acceleration of $1\,\mathrm{m/s^2}$, and that rows or columns of shuttles may move simultaneously when travelling in the same direction. We also assume that each piece of baggage arrives on its own shuttle and leaves the system with it by following the same circular flow as todays baggage trays. Most motions were made to an adjacent cell; for longer paths, multiple transitions in the same direction are combined where possible to avoid multiple acceleration-deceleration cycles and improve overall speed. Collision detection is used to ensure that no physically impossible motions are performed as a consequence of implementation errors (and indeed, none were detected). Table 1 shows the results for maximum- ("max") and average-load days ("avg"). If an input request is made while no input cell is free, the baggage is added to an input queue as stated above; the maximum combined length of all input queues is given in column IQ. The average number of transitions (a good approximation for the travelled distance in meters) for each piece of baggage is contained in column "Steps".

These scenarios contain the minimal number of I/O positions required to achieve the necessary speed. We should note that 4 input and 4 output cells is the lower bound; with only 3 in- or output cells, the speed would be insufficient even if all in- and outputs were to operate at 100% (i.e., no delays for serving new requests) at peak time intervals. This is assuming an effective output time of $5\,\mathrm{s}$ per shuttle, consisting of $2\,\mathrm{s}$ motion onto the output cells and $3\,\mathrm{s}$ removal time.

The naïve implementation already surpasses the current system regarding the space requirement, with 3675 cells of

roughly $1\,\mathrm{m}^2$ each versus $5150\,\mathrm{m}^2$ of the conventional system[3]. A thorough comparison of energy consumption would require the analysis of a specific hardware platform. Nevertheless we can perform a back-of-the-envelope calculation: the current system requires approximately $500\,\mathrm{kWh}$ or $1.8\,\mathrm{GJ}$ per day, largely independent of the amount of stored baggage. When performing 33.6 transitions on the maximum load day with 10040 pieces of baggage, each of the 337344 resulting steps may take $5.335\,\mathrm{kJ}$ to be on par with the conventional system. Assuming a combined weight of shuttle and baggage of 200kg, the required output energy of the engine is approximately $t_{accel} \cdot 200\,\mathrm{kg} \cdot 1\,\mathrm{m/s^2} \approx 0.2\,\mathrm{kJ}$, neglecting friction, per step. Thus, as long as the shuttles provide an engine with an overall efficiency factor of at least 3.7%, energy consumption is also improved.

The combined motion- and destination planning strategies presented yield results far superior to the naïve implementation with a complete road-map. The latter needs 65% more space, 47% more energy (regarding the average load day) and 1-2 additional interconnection points.

Furthermore, the ratio of consumed energy *per stored item* on maximum load days (i.e., design capacity) and average load days is vastly improved: in the layout with aspect ratio 1:10, on an average day we require 86% of the energy consumed on a maximum load day using the naïve implementation for storing a single piece of baggage (i.e., the combined input and output operation). Layout (b) improves this ratio to 74%. Using layout (a), we can further improve it to 62%, thus providing an efficient way to exploit lower-load situations. When accounting for the lower number of I/O operations, the average load day only requires 31% of the energy consumed on the maximum load day, even though we store almost 50% of the goods.

Conclusion

We presented a framework to manage a large fleet of shuttles in a high-density storage environment. To our surprise, the optimization of energy consumption and space requirement are not competing, but go hand in hand: reserving a road-map exclusively for motion purposes increases the distance between input, output and storage locations. We have shown in our simulation that this increase in travelled distance is larger than the energy required to perform evasive motions to retrieve goods without road-map access. The small difference in performance, space requirement and energy consumption between the two aspect ratios demonstrates the robustness of our approach regarding changes in layout.

In comparing our approach with a road-map based implementation, we have shown an improvement not only regarding the existing system, but also a benefit over conventional shuttle-based systems. The benefit of reduced space requirement, which partly offsets higher investment costs for more shuttles, combined with the improved energy requirement, shows a clear advantage of storing each good on its own shuttle.

[3]While additional (I/O-)equipment is not included in this figure, the size of the conventional system, too, is given as a net value. The gross size is approximately $9500\,\mathrm{m}^2$.

References

Ben-Shahar, O., and Rivlin, E. 1998. Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation* 14(4):549–565.

Bylander, T. 1991. Complexity results for planning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, 274–279. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

de Wilde, B.; ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: Cooperative multi-agent path planning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, 87–94. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Dogar, M., and Srinivasa, S. 2011. A framework for push-grasping in clutter. In *Proceedings of Robotics: Science and Systems*.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Gue, K. R., and Kim, B. S. 2007. Puzzle-based storage systems. *Naval Research Logistics* 54:556–567.

Hopcroft, J. E.; Schwartz, J. T.; and Sharir, M. 1984. On the complexity of motion planning for multiple independent objects; PSPACE hardness of the warehouseman's problem. *The International Journal of Robotics Research* 3(4):76–88.

Lindzey, L.; Knepper, R. A.; Choset, H.; and Srinivasa, S. S. 2014. The feasible transition graph: Encoding topology and manipulation constraints for multirobot push-planning. In *Algorithmic Foundations of Robotics XI*, 301–318.

Luna, R., and Bekris, K. E. 2011. Efficient and complete centralized multi-robot path planning. In Borrajo, D.; Likhachev, M.; and López, C. L., eds., *SOCS*. AAAI Press.

Parsons, D., and Canny, J. 1990. A motion planner for multiple mobile robots. In *International Conference on Robotics and Automation*, 8–13. IEEE.

Peasgood, M.; Clark, C. M.; and McPhee, J. 2008. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*.

Stilman, M.; Nishiwaki, K.; Kagami, S.; and Kuffner, J. 2007. Planning and executing navigation among movable obstacles. *Springer Journal of Advanced Robotics* 21(14):1617–1634.