The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Stable Stacking for the Distributor's Pallet Packing Problem

Martin Schuster Richard Bormann Daniela Steidl Saul Reynolds-Haertle Mike Stilman

***Abstract*— We present a novel algorithm that solves the distributor's pallet packing problem. In contrast to existing algorithms, our method optimizes stack stability in addition to stack volume. Furthermore, our algorithm explicitly handles cases where the construction of homogeneous layers of packages with equal height is impossible due to differences in package heights and quantities. The algorithm is a nested beam search that separately optimizes local and global evaluation criteria. We show successful results on both real world and synthetic data sets, compare our performance to an existing algorithm and demonstrate experimental applications in simulation and on a real palletizing robot.**

I. INTRODUCTION

The problem of mixed palletizing plays an important role in the distribution industry. In grocery, beverage distribution centers and parcel services, boxes of different shapes are packed onto pallets. In order to minimize delivery costs, each pallet must be packed efficiently with respect to maximal volume utilization and inter-layer support, making it stable for safe transport. This paper addresses the distributor's pallet packing problem, where an order contains n different types of rectangularly shaped boxes of known dimensions $l_i \times w_i \times h_i$ for $i = 1..n$. The task is to pack a given number n_i of each type of box on a pallet of a given size $L \times W \times H$. Finding an optimal solution to this problem with respect to maximal volume utilization was proven by [1] to be NP-hard.

The main objective of previous research was the maximization of volume utilization on the pallet. Instead, our algorithm focuses on maintaining stability in addition to volume utilization. We accomplish this by using a nested beam search. An outer search optimizes the global density and stability of the pallet, while an inner search finds the best local position for the next box. In contrast to previous work, our algorithm does not create the solution layer-wise. This allows us to construct a pallet using a large number of different box types even when building layers is not possible. Figure 1 shows a plan generated by our algorithm on real world data from a beverage distribution center. Notice that layers differ in height over the area of the panel.

In section III, we present the assumptions made by our algorithm and describe our method in detail. In section IV, results from running the algorithm on both synthetic and real data are shown and compared to the planner presented in [2]. To prove that the algorithm is applicable to problems in the real world, we used data from an American wholesaler and a beverage distributor. In section V, the algorithm is evaluated and section VII discusses strength and weaknesses of our approach. Furthermore, we demonstrate an application of our method in simulation and on a real palletizing robot as shown in figure 2.

The authors are affiliated with Robotics and Intelligent Machines (RIM) at the Georgia Institute of Technology, Atlanta, Georgia 30332, USA. Email: martin.schuster@gatech.edu, richard.bormann@gatech.edu, steidl@gatech.edu, saulrh@gatech.edu, mstilman@cc.gatech.edu

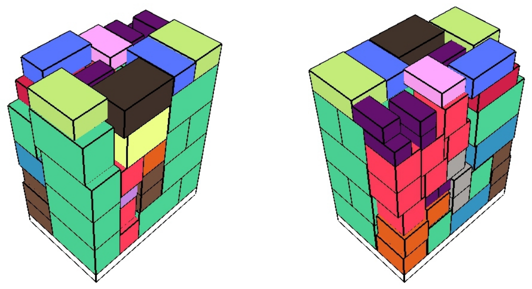
Fig. 1. Simulated result generated by our algorithm for a package list from a beverage distribution center. The pallet is shown from two points of view.

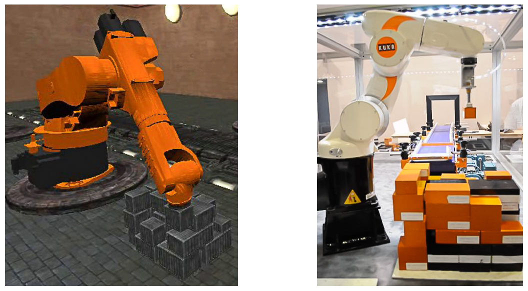
Fig. 2. Algorithm execution in the URSARSim simulator and on the GT/KUKA palletizing cell used for the ICRA-2010 VMAC competition.

II. RELATED WORK

The problem of palletizing is divided into several categories. The manufacturer's pallet loading problem is the simplest form of palletizing. The task is to find a loading pattern for identical boxes for each layer of the pallet [3]. More complex formulations are known as the distributor's pallet packing problem and the multi-pallet loading problem, which both pack non-homogeneous items onto one or several pallets, respectively [4], [5], [6]. This paper addresses the distributor's pallet packing problem. In contrast to previous work, we focus on the stability of the stack in addition to density.

Previous work considers stability mainly for the manufacturer's pallet loading problem. However, the manufacturer's pallet loading problem only contains identical boxes as shown in [3] and [7]. Previous solutions to the distributor's pallet packing problem as [4] take non-identical parcels into account, but pack them only considering the volume utilization and weight constraints [8], not considering stability as a main optimization criterion. In contrast to previous work, our algorithm also focuses on the stability of the pallet while packing non-identical boxes.

In [3] and [7] solutions to the manufacturer's pallet loading problem are presented which create layers of homogeneous items. To maximize the stability of the pallet, adjacent layers are constructed with different loading patterns. [7] defines two criteria for stability, which we also take into account: each box must be supported by two boxes below and a certain percentage of the base must be in contact with boxes below.

[6] gives a solution to the multi-pallet loading problem. For each pallet, it calculates a sub-contingent of the total package list and builds the solution layer-wise. It therefore prefers building homogeneous layers. If this is not possible, heuristics for creating layers with uneven surfaces are provided. Although a stability criterion is mentioned in this paper, the results do not show stable pallets where each box is supported at least by two boxes below.

The distributor's pallet packing problem is addressed in [4], [5] and [2]. They solve the problem for an arbitrary number of box types with the objective to maximize volume utilization of the pallet. The algorithms may coincidentally create interlocks between layers but do not explicitly enforce stability for the entire pallet. [2] constructs layers and sublayers using a heuristic algorithm, that aims to imitate human packing strategies. They solely optimize pallet density, [5] taking additional weight considerations into account.

More advanced solutions are discussed in [9] and [10]. These describe a robot picking one out of six available boxes, packing two pallets at the same time. Unfortunately, no details about these algorithms are released.

III. METHODS

In this section we describe the details of our algorithm. The main objective is to maximize the stability and density of the entire stack. Therefore we designed a nested beam search with an outer, global search presented in section III-G and an inner, local search explained in section III-E. The stack is constructed sequentially; adding a new box creates a new search state. The global search criteria aim for stack density and global stability. The local search guarantees additional constraints and local stability. After presenting the underlying assumptions, the state representation, and the constraints on valid box placements, we will explain the search in detail.

A. Assumptions and Specifications

We define the distributor's three-dimensional pallet-packing problem with the following assumptions:

- Packages are right cuboids with real valued width, depth, height and weight.
- Each package has two distinct orientations: 0° and 90° in the horizontal plane.
- Both the pallet and the packages have constraints on the total weight that each can support.
- Weight is uniformly distributed over the area supporting a package.
- There is a finite, initially known number of packages of each type.
- The algorithm can choose the order of the packages.

We do not orient each box in all six different rotations as done in [2] since the height is usually the shortest dimension in real applications and most palletizing environments are designed for pick and place operations where robots grasp each package on the top surface and place it on its bottom surface, e.g. using vacuum grippers. We do not consider uncertainty in the final box placements due to robot inaccuracy. However, in section VI we use a post processing algorithm to calculate approach paths for a robot arm that do not rely on complete accuracy of previous placements.

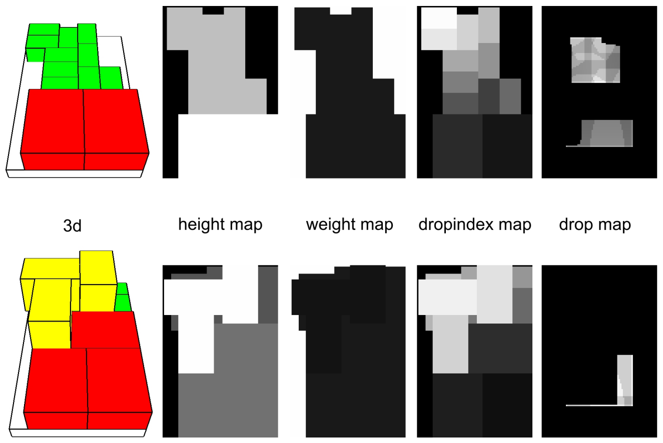

Fig. 3. Feature maps for two different states: Height, weight and dropindex map from state representation and drop map for dropping a red box. Colors range from white to black, indicating the maximum value in the map in white and the lowest value in black.

B. State Representation

In order to facilitate local analysis of the continuous space of object placements we discretize the problem in the horizontal plane. Each search state consists of three feature maps represented by 2D matrices. Each matrix assigns a single number to every cell in a top-down view of the pallet.

The values stored in the feature maps represent:

- *height map*: height of the stack
- *weight map*: maximum weight that can be added
- *drop index map*: index of the topmost box

Examples of maps for two distinct intermediate states can be seen in column 2-4 of figure 3.

We generate a new state by adding a box to the last state and incrementally updating the feature maps for the state.

C. Constraints on Placement

In this section we define hard constraints on the valid placement of a package, referred to as a *drop*. A package can be placed at a specific position on the pallet if all of the following conditions hold true:

- Box placement does not exceed the pallet dimensions.
- Weight of the box does not exceed the maximum weight which can be supported by the boxes underneath.
- Adding the weight of the box does not exceed the maximum weight of the pallet.
- Resulting height of the pallet, after placing the box, does not exceed the maximum height of the pallet.
- At least two opposite edges of the base are supported from below and the box is placed horizontally, i.e. such that the base is parallel to the pallet.

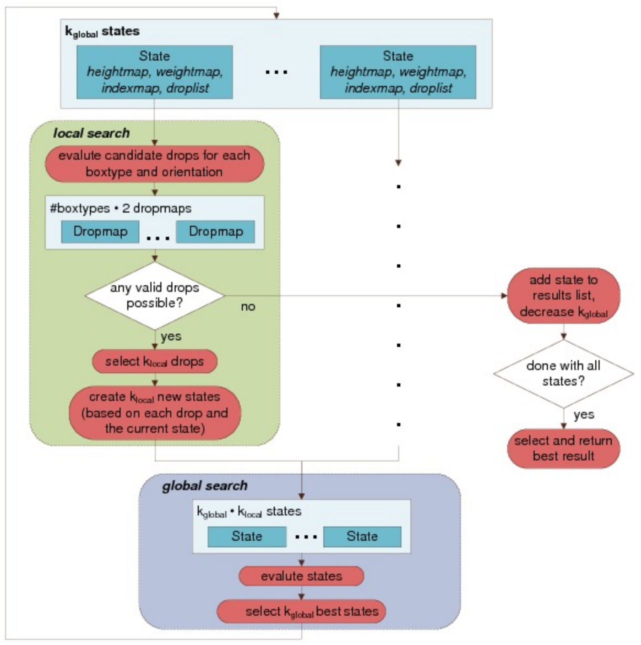
k_global states
State
heightmap, weightmap, indexmap, droplist
...
State
heightmap, weightmap, indexmap, droplist
local search
evalute candidate drops for each boxtype and orientation
#boxtypes • 2 dropmaps
Dropmap
...
Dropmap
any valid drops possible?
no
yes
select k_local drops
create k_local new states (based on each drop and the current state)
add state to results list, decrease k_global
done with all states?
yes
select and return best result
global search
k_global • k_local states
State
...
State
evalute states
select k_global best states

Fig. 4. Planning algorithm overview: combination of global and nested local search

Our weight constraints are similar to the hard constraints defined in [8]. Additional application-specific constraints like the availability of a valid approach path for a robot arm can be enforced as hard constraints for a *drop* when necessary.

D. Search

Having explained the underlying assumptions about the problem, the state representation, and the definition of valid placements, we will now show how the nested beam search optimizes the stability of the stack. The search starts with an empty pallet and then drops packages from above. In order to generate children for each selected state in this search tree, the feature maps for that state are used to search locally for a fixed number of drops. These drops are evaluated only in the context of their local neighborhood, using the criteria explained in section III-F.

We employ a beam search as the outer search over the state space which maintains a constant number of possible stacks. For these stacks we optimize the global evaluation criteria presented in section III-H. Those criteria take the entire pallet into account. The process of selecting and expanding local nodes, referred to as one *iteration of the global search*, is repeated until the algorithm has either placed all available boxes or is unable to place any more packages due to placement constraints. Therefore the maximum search depth of the algorithm corresponds to the maximum number of packages in a search path.

An overview of this algorithm is visualized in figure 4.

E. Local Search

We now present our local search and show that it satisfies the constraints described in section III-C and optimizes for local stability. In order to find suitable candidate drops according to these criteria we evaluate a *dropmap* for each package type and orientation in each state. A *dropmap* is a two-dimensional grid which contains a score value in each cell indicating the utility of dropping the center of the box at that position. This score is calculated as given in section III-F. It also indicates when a drop at this location is impossible due to height, weight or stability constraints. Improper drops are discarded by our algorithm, so that the imposed constraints are enforced. A constant number k_{local} of the best drops of each of these maps are then returned as candidate drops to create a pool of child states for the beam search. These drops are characterized as maxima in the dropmaps. In order to explore the search space in different directions we want to create significantly different candidate drops in one iteration of search. We ensure that by decreasing the values in the dropmaps in the vicinity of a chosen drop. This guarantees that the same or slightly different placements are not selected again in the same iteration due to their low score value.

F. Local Evaluation Criteria

We evaluate each possible drop position for each package according to the following criteria:

- *base supports:* # of boxes that support the base of a box
- *sides supported:* # of sides supported by other boxes
- *surface support area:* percentage of the area of side and base surfaces touching other boxes or the pallet bounding box
- *base height:* $\frac{pallet\ height - box\ stacking\ height}{pallet\ height}$
- *volume:* the volume of the box divided by the volume of the largest box still available for placement

All of these values are normalized to an interval between 0 and 1. For *base supports* we only consider supporting boxes as stable support if they touch the dropped parcel for at least 25% of its base area. Consequently, *base supports* is normalized by division of 4, which is the maximum amount of supporting boxes. A weighted sum of all criteria defines the score in the dropmap. We empirically determined that setting all weights to 1 was the most successful configuration. The *base supports* criterion leads to interlocks between packages which increases the stack stability, while the other criteria are designed to ensure a dense stack. The *volume* criterion prefers the placement of large boxes at the bottom of the stack. The *base height* criterion is intended to prefer stacking at low heights first and filling gaps.

The fifth column in figure 3 shows two examples of drop maps. For each cell, they visualize the value (white: best score, black: no drop possible) of dropping a red package centered on that cell. The lower image indicates that it can be placed on top of the middle of the other red packages.

G. Global Search

In this section we present the method by which global search optimizes for a stable and dense stack. The outer search keeps a list of a constant number of states in memory, later referred to as k_{global}. In each iteration, we generate children for each state from the k_{local} candidate drops that are calculated by the local search as explained in section III-E. From these $k_{global} \times k_{local}$ child states, we select k_{global} new states to explore in the next iteration. Figure 5 depicts

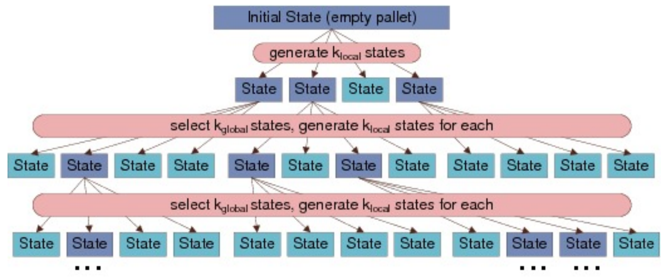
Initial State (empty pallet)
generate k_{local} states
State
select k_{global} states, generate k_{local} states for each
select k_{global} states, generate k_{local} states for each
...

Fig. 5. Beam search schematic

this process. We additionally check for duplicate states using a hashmap and remove them prior to selection. The selection takes the k_{global} best states according to the global evaluation criteria defined in section III-H. The global search iterates until either all available packages are placed or the algorithm cannot find a valid placement for any remaining package according to section III-C.

H. Global Evaluation Criteria

We calculate the scores used in the beam search selection of section III-G as a weighted sum of the following two values, which cause the algorithm to prefer dense and stable stacks:

- *Stack density:* $\frac{\text{Volume of stack}}{\text{Volume of bounding box of stack}}$
- *Average box support:* Average number of packages that each package is placed upon.

The latter criterion favors plans with high interlocks. Taking the average might lead to a high variance, however our local criteria prevent this by choosing plans with high values for *box supports* in each optimization step.

In our experiments, we used the following weights:

$$score_{global} = stack\_density + avg\_box\_support/2.5 \quad (1)$$

The division helps to normalize the average box support, as high values lie around 2.5 and both factors are intended to be weighted equally. The chosen criteria ensure that the most stable solutions proposed by the local search are selected globally with the goal of yielding a stable stack which is densely packed.

I. Iterations

As shown in the previous sections, our algorithm relies on the choice of k_{local} and k_{global}. Empirical evaluation on our data set *Set-S1*, which we will introduce in section IV-C, revealed that results were best when $k = k_{local} = k_{global}$. However, we could not determine a clearly preferable value for k in the range between 1 and 25. Therefore we run the algorithm for $k = 1$ to k_{max} in increments of 5 and select the best result.

IV. EXPERIMENTS

We evaluated our algorithm on two real world and two synthetically created data sets. In this section, we present the experimental setup, describe our evaluation criteria and give details about the data sets. We then discuss our findings in section V. To our best knowledge, we are the first to evaluate an algorithm in this domain both on real wold and synthetically created data sets.

A. Setup

The input to our algorithm is a description of the pallet and a list of package types [11]. The pallet description contains its *width, depth, maximum height* and *maximum weight*. Each entry in the package list consists of *width, depth, height, weight, maximum weight on top* and the *number of packages* of that type. The output is an ordered list of packages and their positions, (X, Y, Z)-coordinates relative to the pallet.

For all experiments we used a planning resolution of 1cm in the horizontal plane and a height tolerance of 1cm because we expect this to be in the range of inaccuracies in robot movements and box dimensions. A higher resolution might reduce gaps between boxes but requires a longer runtime for the algorithm. Unless noted otherwise, experiments were performed on the widely used Europool pallet [12] with a base area of 120cm×80cm and a maximum height of 150cm. The actual size of the pallet is not important. Our algorithm only depends on the ratio of pallet to the package dimensions, which is diverse over the test cases.

B. Evaluation Criteria

We consider five criteria to evaluate the results of our experiments: *number of boxes stacked*, the *stack density* and *average box support* as defined in section III-H, the *average face contacts* and the total algorithm runtime over all iterations as described in section III-I. *Average face contacts* is defined as the average number of packages each package is touching on all faces.

The *stack density* criterion indicates the volume utilization of the pallet. We use the *average box support* and *average face contacts* criteria to evaluate stack stability.

C. Synthetic Data Sets

We tested our algorithm on two different synthetically generated data sets. The first, *Set-S1*, consists of 10 synthetically created package lists. We defined packages with random dimensions from $10cm$ to $55cm$. We ran our algorithm for six iterations with $k_{max} = 25$.

The second artificial data set, *Set-S2*, consists of five randomly created package lists. It was presented in [2] as "Set#1-5" for benchmarking their layer-wise planner. We used the same pallet size of $104cm \times 96cm \times 84cm$, the package dimensions range from $1cm$ to $67cm$. As one list contains more than 1000 packages, we ran our algorithm only with $k_{max} = 5$.

Visualization for stacks generated from package lists from *Set-S1* and *Set-S2* are shown in figure 6.

D. Industrial Data Sets

Next, we performed tests on real world data sets from two major American companies. *Set-R1* was obtained from a beverage distributor and contains 278 package lists. The package dimensions range from $2cm$ to $49cm$, the number of package types from 2 to 20. *Set-R2* consists of 8 package lists from a large wholesale distribution center with 10 to 38 package types having dimensions from $3cm$ to $51cm$. A high percentage of the package types, 57% for *Set-R1* and 98% for *Set-R2*, have at most four boxes per pallet.

TABLE I
AVERAGE RESULTS AND STANDARD DEVIATION (STD) FOR SYNTHETIC DATASETS *Set-S1*, *Set-S2* AND REAL WORLD DATASETS *Set-R1*, *Set-R2* FOR OUR ALGORITHM AND BALTACIOGLU [2]

package lists	**# package types**	**# packages stacked**	**stack density**	**avg. box support**	**avg. face contacts**	**runtime in** s	**max. runtime in** s
Set-S1	5.30	42.1	**77%**	**1.82**	6.56	681s	940s
std	1.34	13.52	7.36%	0.31	0.44	189s	
Set-S1 [2]	5.30	59.6	90%	1.04	6.27	2.3s	20s
std	1.34	20.49	7.29%	0.04	0.84	6.2s	
Set-S2	14.60	584	**89%**	**1.92**	8.88	71s	139s
std	11.26	503	4.53%	0.54	2.32	56s	
Set-S2 [2]	14.60	839	89%	1.14	6.30	14s	41s
std	11.26	741	11.74%	0.22	2.21	17s	
Set-R1	9.17	44.10	**70%**	**1.54**	5.84	196s	537s
std	5.12	17.87	7.94%	0.25	0.93	125s	
Set-R1 [2]	9.17	59.72	85%	1.07	5.94	0.4s	6s
std	5.12	38.40	13.40%	0.08	1.90	0.6s	
Set-R2	26.13	35.38	**68%**	**1.44**	6.34	460s	854s
std	10.82	16.14	4.01%	0.17	0.98	283s	
Set-R2 [2]	26.13	35.38	56%	1.03	5.18	0.1s	1s
std	10.82	16.14	9.10%	0.08	1.15	0.4s	

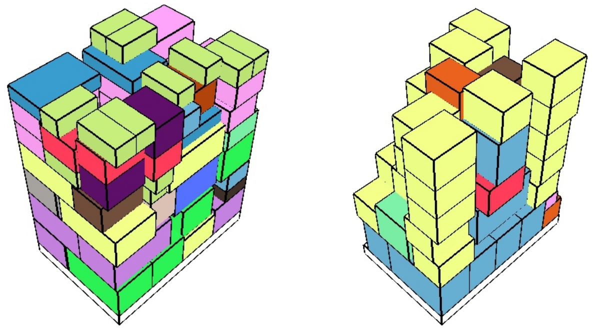
Fig. 7. Plans for two package lists from *Set-R1*

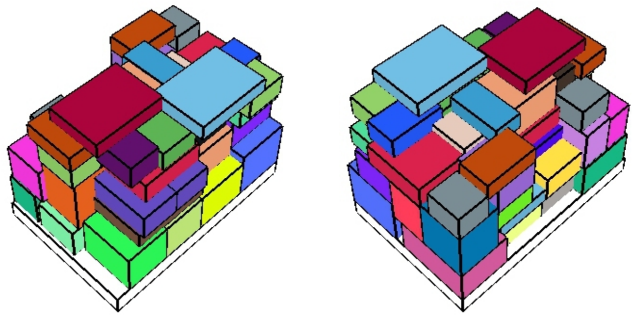
Fig. 8. Plan for one package list from *Set-R2*, shown from both sides

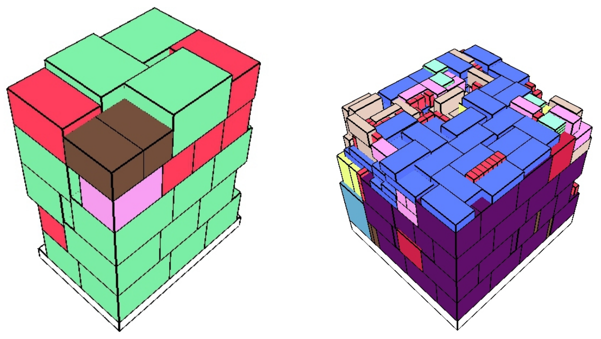
Fig. 6. Plans for list from *Set-S1* (left) and *Set-S2* (right)

For *Set-R1* we ran our algorithm for three iterations up to a maximum number of $k_{max} = 10$ search-nodes per level. For the smaller *Set-R2* we used four iterations up to $k_{max} = 15$. We present the average results in table I and show visualizations for our plans in figure 7 and figure 8.

V. DISCUSSION

A. Quality

1) Synthetic Data Sets: On the generated data set *Set-S1* our algorithm achieved an average density of 77%, having a mean of 1.82 packages supporting each one from underneath. These high values were reached because the package lists contained only 4 to 9 different package types. This simplified the generation of sub-stacks with similar height onto which further boxes can be placed.

For the second synthetic data set, *Set-S2*, our algorithm was able to achieve a volume utilization of 89% on average. Such high density values can be achieved due to the availability of unrealistic small packages that allow the planners to fill gaps.

2) Industrial Data Sets: The greater variation in real world package dimensions, especially heights, makes the task more difficult than handling synthetic data. This results in lower average values for *density* and *box support*, presented in table I. Our algorithm is still able to build dense and stable stacks by combining a large number of different package types as shown in figures 1, 7 and 8.

3) Comparison to Baltacioglu [2]: We performed a direct comparison with the planner presented in [2], using all of our data sets and evaluation criteria. The results are presented in table I. Our higher values for *average box support* indicate the stability of our stacks. Values greater than 1.5 indicate that at least every other package is stabilized by two packages below. The resulting interlocks ensure that packages do not fall off the stack when the pallet is moved or lifted. In contrast to that, the values for [2] are below 1.15 for all test cases. This results in pallets that are not as stable or safe for delivery because most packages are only supported by one other package from below, as can be seen in the visual comparison in figure 9. Although there is a visible tradeoff of density vs. stability, our density values are equal to or higher than [2] for two out of four test sets. The lower variance in stack density also shows that we constantly produce a dense stack while maintaining stability.

4) Challenges: During development, we observed two significant challenges. First we aim to maximize the stability of the stack by placing boxes directly next to each other without any gaps. However this may result in gaps near the opposite corner of the first package placement. If those gaps are created in the lower part of the stack, they steadily reduce

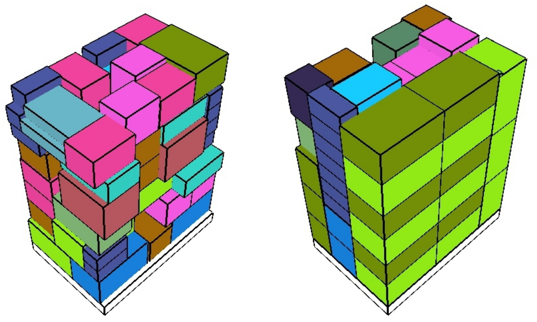

Fig. 9. Comparison: One package list from set *Set-R1*: Our algorithm (left) and Baltacioglu [2] (right)

the available area as the height of stack increases.

In some cases, our algorithm produces problematic high stacks of packages of a single type. For instance, consider the right image in figure 7. We plan to introduce additional constraints on placement to directly address this problem.

B. Runtime

The maximum runtime of our algorithm depends on the *discretization resolution*, the *pallet size*, the *number of package types* and k_{max}, which gives the number of iterations. By updating the feature and drop maps only in the vicinity of the last drop, we gained significant performance increases compared to a recalculation of the entire map.

Table I shows the runtimes for all test sets. Every plan presented took less than 10min to generate on a standard laptop with an Intel Core 2 Duo 2.5 Ghz. Although our algorithm is significantly slower than [2], it is fast enough for real world applications. The algorithm is also well suited for future parallelization because the generation of each state is independent of other states at the same level.

VI. APPLICATION

We performed preliminary tests in simulation and on a real robot. Both tests show that the output of our algorithm can be used to build a stable pallet with a real robot arm, as shown in figure 2.

A. Execution in Simulation

For simulation we used the USARSim simulator (Unified System for Automation and Robot Simulation) which is a high-fidelity dynamic simulation of robots and environments based on the Unreal Tournament game engine. In order to handle uncertainty of robot placements, we used a post-processing algorithm to calculate approach paths for placing a package onto the pallet. Based on these, the robot arm approached each package placement position with a small lateral velocity in both x and y. This ensured that small errors in previous package placements are corrected by the current package.

B. Execution on Palletizing Robot

In addition to simulation in USARSim, the algorithm was successfully executed on the GT/KUKA palletizing cell used for the ICRA-2010 VMAC competition. The cell included a conveyor, a vacuum-gripper and a pallet. The execution on a robot arm shows that our algorithm is applicable to real palletizing systems.

VII. CONCLUSIONS AND FUTURE WORK

Our new method is a promising approach to solving the distributor's pallet packing problem while maintaining stability of the stack. We showed how the inner search guarantees constraints and ensures local stability. By evaluating global criteria, the outer search aims for density and stability of the entire stack. We showed the success of our algorithm by evaluating it on real-world data, comparing it to another planner and executing plans in simulation and on a real robot.

The performance of our algorithm depends heavily on the definition of appropriate local and global evaluation criteria for the searches. A thorough analysis of the effect of single criteria as well as a search for additional criteria to cope with particular problematic cases is subject to future research. In real applications, our planner could be used in combination with a layer-wise algorithm, handling only the part of the pallet the layer-wise planner cannot generate layers for.

All the data sets used in this work are available at [11]. We encourage future researchers to compete with our results and share the details of their algorithms to further improve the performance and efficiency of solutions to the practical challenge of mixed palletizing.

VIII. ACKNOWLEDGMENTS

We thank Henrik Christensen for his support in defining the goals and gathering data for this research. This work was initiated in the course *Robot Intelligence: Planning* at the Georgia Institute of Technology.

REFERENCES

[1] H. Yaman and A. Şen, "Manufacturer's mixed pallet design problem." European Journal of Operational Research, 2008.

[2] E. Baltacioglu, "The distributer's three-dimensional pallet-packing problem: A human intelligence-based heuristic approach," 2001.

[3] W. Kocjan and K. Holmström, "Generating stable loading patterns for pallet loading problems." The Fifth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR08, 2008.

[4] E. Bischoff, F. Janetz, and M. Ratcliff, "Loading pallets with non-identical items." European Journal of Operational Research 84, 1995.

[5] E. E. Bischoff, "Three-dimensional packing of items with limited load bearing strength," *European Journal of Operational Research*, vol. 168, no. 3, pp. 952–966, 2006.

[6] J. Terno, G. Scheithauer, U. Sommerwei, and J. Riehme, "An efficient approach for the multi-pallet loading problem," *European Journal of Operational Research*, vol. 123, pp. 372–381, 1997.

[7] E. Bischoff, "Stability aspects of pallet loading." OR Spectrum Vol. 13, Springer-Verlag, Dec. 4 1991.

[8] M. Ratcliff and E. Bischoff, "Allowing for weight considerations in container loading," *Operations Research Spektrum*, vol. 20, pp. 65–71, 1998.

[9] C. Wurll and B. Schnoor, ""robot picking system" automated order picking with industrial robots." IASTED International Conference on Robotics and Applications, 2003.

[10] ——, "Robot picking system: Automated order picking with industrial robots," in *Robotics and Applications*. IASTED/ACTA Press.

[11] M. Schuster, R. Bormann, D. Steidl, S. Reynolds-Haertle, and M. Stilman. (2010) Mixed palletizing data. [Online]. Available: http://www.golems.org/node/1097

[12] "DIN EN 13698-2:2009-10: Pallet production specification - part 2: Construction specification for 1000 mm x 1200 mm flat wooden pallets."