Pushing and Pulling

Computing push plans for disk-shaped robots,
and dynamic labelings for moving points

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 22 augustus 2013 om 16:00 uur

door

Dirk Henricus Paulus Gerrits

geboren te Geldrop

Dit proefschrift is goedgekeurd door de promotiecommissie:

voorzitter:	prof.dr. E.H.L. Aarts
1^{e} promotor:	prof.dr. M.T. de Berg
copromotor:	dr. K.A. Buchin
leden:	prof.dr. M.J. van Kreveld (UU)
	prof.dr. D. Halperin (Tel Aviv University)
	dr. N. Bansal
	Prof.Dr. A. Wolff (Universität Würzburg)

A catalogue record is available from the
Eindhoven University of Technology Library
ISBN: 978-90-386-3421-0

Colofon

This thesis and its related source code have been written using the GNU Emacs text editor, mostly under the Linux-based Ubuntu operating system, partly in Microsoft's Windows 7 operating system. It was typeset using LaTeX. The experiments of Chapter 5 were coded in C++ with the Qt library and the GNU Compiler Collection. Their results were graphed using the ggplot2 package for the R environment for statistics. All other figures in this thesis were prepared with the Ipe extensible drawing editor.

The work in the thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Cover design: Gijs Gerrits (concept), Robin Wouters (graphics), Tommy Wouters (layout)

Photography: Paul Gerrits Fotografie, Bergeijk

Printing: Ipskamp Drukkers, Enschede

Acknowledgments

The last few years of my life have been quite challenging, even outside of research. I don't know how I would have made it through without the support of my loved ones. I especially want to thank my girlfriend, Marleen Tubben, and our cat, Joy, for the much-needed warm welcome after a long day. My parents, Paul Gerrits and Joke Gerrits-Wouters, were also always there for me when I needed them, as were my younger brother, Gijs Gerrits, and Marleen's parents, Wim Tubben and Annie Tubben-Stolte. They've all been a great support. For this thesis in particular, my father made the portrait on page 123, and my brother came up with the design for the cover, which was then fleshed out by my cousins Robin and Tommy Wouters.

Research-wise, I owe everything to my supervisors Alexander Wolff (during the first year of my PhD) and Mark de Berg (during my Master's and the remainder of my PhD). This thesis would not have been written if it had not been for them. Mark especially has always had his door open for me. One could always come to his office with any problem, both inside or outside of academia, and come out feeling much better afterwards. More than just a supervisor, he has been an inspiration, and a true friend.

Apart from my supervisors, I also want to thank the many other colleagues I've had over the years, who have all contributed to made the workplace a happy one. During my time here, our research group's staff has consisted of various subsets of Mark de Berg, Kevin and Maike Buchin, Peter Hachenberger, Herman Haverkort, Bettina Speckmann, and Alexander Wolff. I want to thank Alexander and Peter in particular for organizing various nice field trips and bowling nights, and Bettina and Kevin for many pleasant conversations during our commutes by train. Those with Kevin would eventually lead to the results in Chapter 3. All of them are responsible for providing a friendly environment that fosters excellence in conducting and presenting research. Whenever there's a computational geometry talk given by someone from Eindhoven, you know it will be worth attending.

It is always nice to be able to share your experiences with others, so it is wonderful to have been among so many fellow PhD students. Special thanks go to Chris Gray and Elena Mumford, with whom I shared an office while work-

ing on my Master thesis and during the start of my PhD research. They have since graduated and moved on to pursue other opportunities, as have Mohammad Abam, Jarosław Byrka, Matthias Mnich, Nataša Jovanović, Constantinos Tsirogiannis, Amirali Khosravi, and Kevin Verbeek. Special thanks also to Marcel Roeloffzen, Wouter Meulemans, and Arthur van Goethem, for the many enjoyable chats about video games, and for making trips to conferences an enjoyable experience. They are yet to graduate at the time of this writing, as are Anne Driemel, Ali and Saeed Mehrabi, and Farnaz Sheikhi. I wish all of them the best of luck with their own PhD theses, although I'm sure they'll succeed without the need for luck.

As I am not very good with names, I won't attempt to list the dozens of Master students who have been a part of our research group at some point or another during my own stay. The risk of me making an error or omitting someone is simply too great. I do want to thank all of them, for stimulating lunch-break conversations and contributing nice research talks to our seminars.

While still in the process of writing this thesis, getting it printed, arranging the defense, and so forth, I had already moved on to a new and completely different research project. I want to thank Arianna Betti, as well as her colleagues Jeroen Smid and Hein van den Berg, for giving me this opportunity, and being my guides in the digital humanities. They have been most patient with me whenever I had to take care of things related to my thesis work. I shall do my utmost best to make up for lost time after my defense.

Outside of the workplace, I want to thank Peter Kooijmans, Floris Snijders, Stefanie Hasler, Isak Rydén, Oskar Skoog, Emil Styrke, Erik Pettersson, and Henrik Abelsson, for letting me blow off some steam through online games and conversations. Their friendship has been invaluable.

Last but not least, I want to thank Erik Torstensson, for enriching the lives of anyone who knew him. In life he was like an older brother to me. Much of my thinking about life, programming, and computer science is shaped by the many cheerful conversations I've had with Erik. To this date, hardly a day goes by where I don't think back fondly to them. Whenever I have the urge to name a variable in my programs `mat`, I can hear him pointing out the silliness of cryptic abbreviations in an age of wide-screen monitors and TAB-completion. I then smile and change the name to `rotationMatrix`. You are sorely missed, old buddy. I dedicate my PhD thesis to you.

In memory of

Erik Torstensson

6 June 1982 – 23 May 2009

Contents

Introduction 1

This thesis covers two classes of problems that seem unrelated at the surface.

The first involves *path planning* [51, 53], a fundamental problem in robotics where the objective is to find a way to move a robot to its destination, without bumping into obstacles along the way. *Manipulation planning* [61] is a variant of this problem, in which the robot is to move a passive object to a certain destination, rather than to move there itself. In Chapter 2 of this thesis we study a manipulation-planning problem where a disk-shaped robot is *pushing* [61] a disk-shaped object through an environment of polygonal obstacles.

The second class of problems we study involves *map labeling*, which is the task of placing textual *labels* with features on a map such as cities (points), roads (polygonal paths), and lakes (polygons). For readability, labels should not overlap, which means that one should carefully choose which labels to place and where to place them. It takes cartographers considerable time to do this by hand [66], and for some applications it is not even possible to do by hand ahead of time. In air traffic control, for example, a set of moving points (airplanes) has to be labeled at all times, each *pulling* its label along behind it [25]. In interactive maps users may pan, rotate, and/or zoom their view of the map, which may also require relabeling. While there is a vast body of literature on algorithmic methods for the kind of static label placement needed in cartography, far fewer results are known for the kind of dynamic label placement problems inherent in air traffic control and interactive maps. This is the topic of Chapters 3–5 of this thesis.

While static labeling is certainly very different from robot motion planning, when considering dynamic labeling the connection is easily made. Instead of the labels being pulled by their points, we can interpret them as pushing the points along. Labeling moving points then becomes a multi-robot variant of the pushing problem. Indeed, in the chapters on dynamic labeling we even reuse some of the machinery established in the chapter on computing robot push plans. For the most part, however, it makes more sense to view the two problem

classes in their own respective contexts, as we shall do for the remainder of this introduction and the thesis as a whole.

In these discussions we shall talk of algorithms, their performance in the real RAM model using "big-O notation", as well as their approximation ratios and the computational complexity of the problems they solve. Readers not familiar with these standard notions from theoretical computer science may refer to textbooks such as *Introduction to Algorithms* [22].

1.1 Manipulation planning

In many applications of robotics, especially in industrial settings, a robot needs to *manipulate* objects other than itself. Often this involves *prehensile* manipulation, where the robot *grasps* [61] the object. This allows the manipulation task to be decoupled into stages, for which motion planning can be done independently. First the robot picks up the object, then it moves to another location, and finally it puts down the object.

Non-prehensile manipulation is more complicated, but more flexible and sometimes more efficient. Instead of carrying tools and small parts around, robots could simply throw them between each other across the factory [63]. Non-prehensile manipulation also helps when an object is too large, or too heavy to be grasped, or when the robot lacks a grasping mechanism. We could then try rolling the object [4], for example, or squeezing it [36]. *Pushing* the object is a particularly useful form of non-prehensile manipulation in this case.

1.1.1 Pushing without obstacles

The way an object moves in response to being pushed can be quite complex, depending on the shapes and frictional distributions of the pusher and the object [62]. In a simple setting, such as a uniform-density disk being pushed by another disk, the analysis is not so difficult. Igarashi et al. [42] present a simple method for this case, based on defining a conceptual magnetic field around the object and moving the robot as if it were attracted by this field. With more general shapes, however, the behavior of the object in response to the pusher seems hard to predict analytically. And yet, children develop an intuition for this task at a fairly young age. Such *unsupervised learning* for the pushing task has been mimicked in robotics research.

Yoshikawa and Kurisu [96] and Lynch [58] present some early approaches to learning the friction characteristics of an object through exploratory pushes. Salganicoff et al. [85] apply these ideas to actually pushing an object to a desired location. They consider a notched object of arbitrary shape lying flat on a plane. A robotic "finger" then prods the object at the notch in varying directions, with the notch ensuring a fixed point of contact throughout. The effect of these prods on the object are monitored. This yields a sequence of (push motion, object

motion)-pairs, representing what the robot has "learned" about pushing this object. Through these pairs a curve is fitted, yielding an approximate function from desired object motions to the needed push motions. This function is then used to plan the needed pushing motion to get the object to its destination. Additional learning is done while pushing, to further correct the pushing motion along the way.

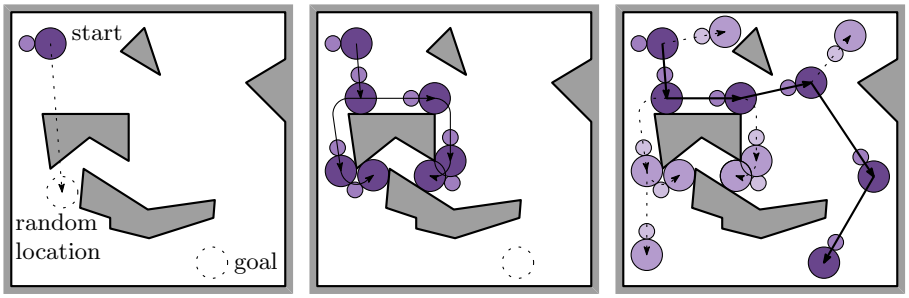
This early work has since seen a sequence of improvements. Walker and Salisbury [93] lift the restriction of a fixed point of contact. They consider a disk-shaped robot which tries pushing the object not just from varying angles, but also from various points of contact along the object's boundary. Lau et al. [52] store the learned information of such a disk-shaped robot in a different way, so that knowledge learned about pushing an object can also be applied to pushing previously unencountered objects of the same shape but a different size. In the method of Kopicki et al. [49], pushing knowledge can even be applied to unencountered objects of different shapes. Furthermore, their method can deal with 3-dimensional objects pushed by a ball-shaped robot. The method of Zito et al. [97] has similar properties, but learns from a physics simulator instead of from actual pushing. It can thus perform mental trial-and-error before physically pushing the object. (This of course presumes an accurate simulation of the object's physical characteristics, which may still require some physical, exploratory pushes to develop.)

1.1.2 Pushing among obstacles

Once the robot knows how an object will respond to its pushing motions, it can try to tackle the harder task of having to avoid obstacles while pushing. Lau et al. [52] propose using a standard path planner to determine a high-level path for the object that avoids the obstacles. A few "waypoints" are then selected along this path, and one of the above "obstacle oblivious" pushing methods is used to push the object between successive waypoints. While this method works in some simple cases, it may fail in others.

Agarwal et al. [2] take a more direct approach to deal with obstacles in push planning. They look at the case of a point-sized pusher and a unit-disk object. Their algorithm discretizes this pushing problem in two ways: the angle at which the pusher can push is constrained to $1/\varepsilon$ different values, and the combined boundary of the polygonal obstacles is sampled at m locations to give potential intermediate positions for the object. The algorithm then runs in $O((1/\varepsilon)m(m+n)\log n)$ time, where n is the total number of obstacle vertices. It is only guaranteed to find a solution if $1/\varepsilon$ and m are large enough. Moreover, it assumes that the pusher can get to any position around the object at all times. While this is true for their point-size pusher, this is not the case for a pusher of non-zero size: there may be obstacles in the way.

Nieuwenhuisen et al. [69] propose a more robust way to deal with obstacles in push planning. Instead of trying to avoid the obstacles entirely, they allow



(a) (b) (c)

Figure 1.1: Illustration of the algorithm by Nieuwenhuisen et al. [69] to push the larger, darker *object* disk using the smaller, lighter *pusher* disk.

the object to be pushed along the obstacle boundaries, which then act as a guide for the object's motion. Their algorithm works for a disk-shaped object and a disk-shaped pusher, and is inspired by the *Rapidly-exploring Random Trees (RRT)* path-planning algorithm of LaValle and Kuffner [54]. It works by incrementally building a tree of reachable *configurations*, which specify a position of both the object and the pusher. The tree's root corresponds to their initial positions. A random object location o' is picked, the configuration (o, p) in the tree is determined for which o is nearest to o', and a path τ from o to o' is computed that would be plausible in the absence of obstacles (Figure 1.1(a)). A subroutine then computes a path σ for the pusher that takes the object as far along τ as is possible without going through obstacles. The resulting configuration (o'', p''), ideally with $o'' = o'$, is then added to the tree. The edge $(o, p) \to (o'', p'')$, annotated with the paths τ and σ, connects it to the rest of the tree. If $o' \neq o''$, then either the object hit an obstacle along the way, or there was an obstacle that the pusher could not avoid. In this case the subroutine is called again with paths to push the object in clockwise and counterclockwise directions around this obstacle, yielding additional edges (Figure 1.1(b)). Once in a while, instead of choosing o' randomly, the object's destination location is chosen. Assuming an obstacle-avoiding push plan exists at all, this process will eventually succeed, with a probability going to 1 as the computation time goes to infinity (Figure 1.1(c)).

1.1.3 Our results

Recall the method of Nieuwenhuisen et al. [69] mentioned above for pushing a single disk-shaped object with a single disk-shaped pusher. It repeatedly generates a candidate path τ for the object, and passes it to a subroutine that computes to what extent the path can be realized by a pusher while avoiding

the obstacles. Much of the efficiency and effectiveness of the algorithm depends on the method used to implement this subroutine.

Nieuwenhuisen et al. themselves implement the subroutine using an $O(n^2)$-space data structure, which is created beforehand in $O(n^2 \log n)$ time. Their subroutine then runs in $O(kn \log n)$ time when τ consists of k line segments and circular arcs. This latter bound is pessimistic, and assumes the obstacles are long, skinny, and tightly packed. If one assumes low obstacle density (a notion to be made precise in Section 2.1.4), then this bound can be improved to $O((k+n)\log(k+n))$. The preprocessing time and space remain quadratic, however. Neither for high nor low obstacle density does their algorithm guarantee that the constructed push plan is optimal in any way.

In Chapter 2 we present a new method to implement this subroutine, that improves on the method of Nieuwenhuisen et al. in several ways. Our method does not need $O(n^2 \log n)$ time and $O(n^2)$ space for preprocessing the obstacles, and can handle more general classes of object paths than just sequences of line segments and circular arcs. Moreover, if there exists a *contact-preserving push plan*, in which the pusher maintains contact with the object at all times, then we compute the shortest such push plan. As we will show in Section 2.4, however, sometimes the pusher must let go of the object on occasion to resume pushing from a different location. We can also compute such *unrestricted push plans*, although we can then not guarantee finding the shortest one. The running times of our method are summarized in Table 1.1. These results have been accepted for publication in the *International Journal of Computational Geometry & Applications* [12].

	High obstacle density		Low obstacle density	
	NH et al.	Our method	NH et al.	Our method
Preprocessing	$n^2 \log n$	$n \log n$ $^{(*)}$	$n^2 \log n$	$n \log n$ $^{(*)}$
Any CPPP	$kn \log n$	$kn \log n$ $^{(*)}$	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$
Shortest CPPP	—	$kn \log n$ $^{(*)}$	—	$kn \log n$ $^{(*)}$
Any UPP	—	$kn \log k + kn^2 \log n$	—	$(k+n)\log(k+n) + kn$

$^{(*)}$ These entries are expected times. For the worst-case times, replace $\log n$ by $\log^2 n$.

Table 1.1: The asymptotic running times of the subroutine of Nieuwenhuisen et al. compared to ours for computing contact-preserving push plans (CPPP) and unrestricted push plans (UPP) for a given object path.

1.2 Map labeling

Map labeling (see Figure 1.2) involves associating textual *labels* with certain *features* on a map such as cities (points), roads (polygonal paths), and lakes (polygons). This task takes considerable time to do manually [66], and for some applications *cannot* be done manually beforehand. In air traffic control, for example, a set of moving points (airplanes) has to be labeled at all times. In interactive maps users may pan, rotate, and/or zoom their view of the map, which typically requires relabeling. It is therefore unsurprising that map labeling has attracted considerable algorithmic research. The online Map Labeling Bibliography [94], for instance, lists 371 references published up to 2009. On the practical side this research has produced heuristic algorithms and their experimental evaluation. On the theoretical side it has led to hardness proofs and algorithms with guaranteed approximation ratios. We shall review some of these results for the labeling of point features.

1.2.1 Label models

A good labeling for a point set has readable labels, and an unambiguous association between the labels and the points. The readability requirement has been formalized by regarding the labels as axis-aligned rectangles slightly larger

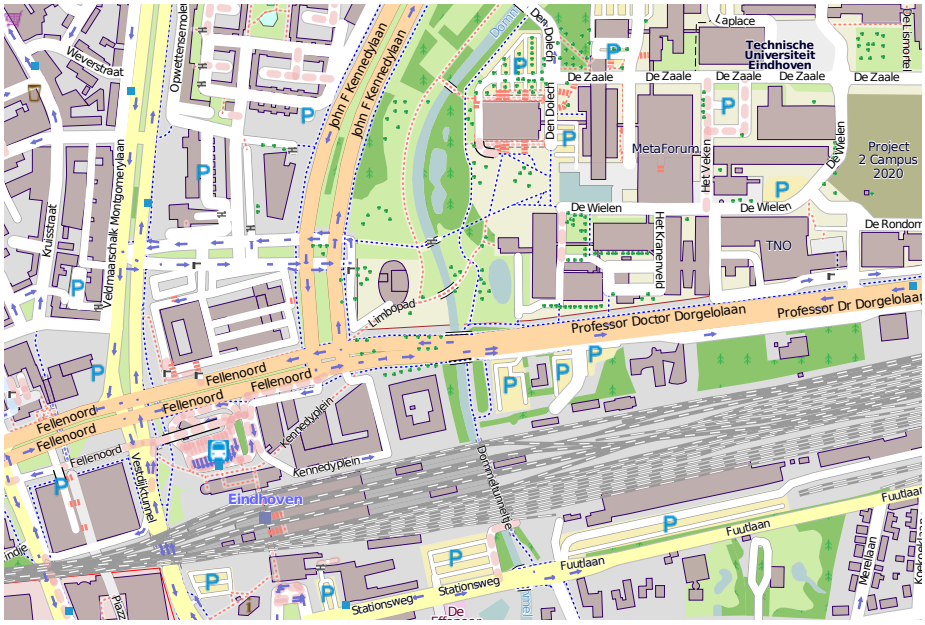


Figure 1.2: An example of a labeled map from OpenStreetMap.org.

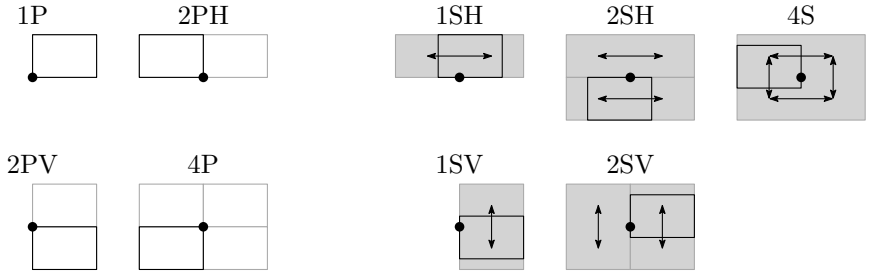


Figure 1.3: The fixed-position (left) and slider models (right) for point labeling.

than the text they contain, which must be placed without overlap. To associate such labels clearly with their points, one option is draw a line segment or curve (called a *leader*) between the point and its label. We shall focus on another option, however, in which the labels are placed so that they contain their point on their boundary. Since not all such label placements are equally desirable [44], various *label models* have been proposed which then specify the subset of allowed positions for the labels.

In the *fixed-position models*, every point has a finite number of *label candidates* (often 4 or 8), each being a rectangle having the point on its boundary. In particular, in the 1-position model one designated corner of the label must coincide with the point. In the 2-position models there is a choice between two adjacent corners, and the 4-position model allows any corner of the label to coincide with the point. These models are illustrated in the upper-left 2x2 block in Figure 1.3, where we use the following notation: 1P denotes the 1-position model, 2PH and 2PV denote 2-position models (where the H and V indicate whether the two designated corners are endpoints of the same horizontal or vertical edge), and 4P denotes the 4-position model.

The *slider models* introduced by Van Kreveld et al. [50] generalize the fixed-position models. In the 1-slider (1SH, 1SV) models one side of the label is designated, but the label may contain the point anywhere on this side. In the 2-slider (2SH, 2SV) models there is a choice between two opposite sides of the label, and in the 4-slider (4S) model the label can have the point anywhere on its boundary (see the last row and column in Figure 1.3).

1.2.2 Static point labeling

Ideally, one would like to place labels from the label model for all points, with the interiors of the labels being pairwise disjoint. Unfortunately, this is not always possible. Moreover, the *decision problem* of determining whether this is the case is strongly NP-complete for the 3-position [45], 4-position [31, 60], and 4-slider [50] models. This is true even if all label rectangles are of the

same dimensions, in which case they can all be assumed to be unit squares by applying a suitable scaling transformation.

When we want to compute a labeling for a point set we may choose to deal with this difficulty in several ways. Firstly, we may reduce the font size, that is, shrink the labels, until all labels fit. The *size-maximization problem* asks to label all points with pairwise non-intersecting labels of maximal size. Alternatively, we may remove some labels. The *(weighted) number-maximization problem* asks to label a maximum-cardinality (maximum-weight) subset of the points with pairwise non-intersecting labels of given dimensions. These and similar optimization problems are strongly NP-hard for the 3-position, 4-position, and 4-slider models, as computing optimal solutions for them would solve the aforementioned decision problem. Additional results about their complexities are also known, however, as discussed next.

Size maximization. Size maximization of identical square labels in the 4-position model does not even admit a polynomial-time $(2-\varepsilon)$-approximation algorithm for $\varepsilon > 0$, unless P = NP [31]. The 2-approximation algorithm of Formann and Wagner [31] is therefore the best one can hope for, in theory. In practice, their method was found to perform poorly, and an alternative 2-approximation by Wagner and Wolff [92] achieves far better results. Qin and Zhu [80] present yet another such 2-approximation algorithm, this time for the 4-slider model. Jiang et al. [46] achieve an approximation ratio of slightly less than 3 for a model with circular labels that can be rotated around their points.

Number maximization in theory. Number maximization is strongly NP-hard with unit-square labels even in the 1-position model [32, 43]. It is, however, easier to approximate than the size-maximization problem. For unit-height labels there exists a polynomial-time approximation scheme (PTAS) in all fixed-position and slider models, both for weighted and unweighted points.

For unweighted points, with unit-height labels in the fixed-position models, Agarwal et al. [1] give the first such PTAS, as well as an $O(n \log n)$-time 2-approximation algorithm. Chan [18] later improved the running time of their PTAS. Van Kreveld et al. [50] obtain the same results as Agarwal et al. for the slider models. For labels that are arbitrary rectangles, Chalermsook and Chuzhoy [17] achieve an $O(\log \log n)$-approximation for the fixed-position models. Whether this more general case admits a PTAS remains an open problem. There are also not yet any approximation algorithms for the slider models with arbitrary rectangular labels.

For weighted points the situation is somewhat similar. For unit-height labels in the fixed-position models, Poon et al. [79] show how to obtain a PTAS, as well as an $O(n \log n)$-time 2-approximation algorithm. They also obtain a $O(n^2/\varepsilon)$-time $(2+\varepsilon)$-approximation for the slider models. The running time of their PTAS was later improved by Chan [18]. A PTAS for the slider models remained elusive for some time, but was finally achieved by Erlebach et al. [30],

through complicated graph-theoretical methods. For labels that are arbitrary rectangles, Berman et al. [14] achieved a $\lfloor 1 + \log_k n \rfloor$-approximation for the fixed-position models, for any integer $k > 1$. Chan and Har-Peled [19] later managed to obtain a slightly better approximation ratio of $O(\log n / \log \log n)$. Again, whether arbitrary rectangles in the slider models admit similar results, and whether there is a PTAS for it, are unknown.

Number maximization in practice. Apart from the above methods that work well in theory, plenty of work has gone into developing and evaluating various method in practice. Christensen et al. [20] give an early overview of such *heuristic* algorithms. They experimentally compare a greedy method, a force-based method, as well as methods based on gradient ascent, simulated annealing, and integer linear programming. In their study simulated annealing produced the best results. Simulated annealing, however, is just one of many so-called *metaheuristics* [57]. Others that have since been applied to point labeling include evolutionary algorithms, tabu search, and ant colony optimization.

The 2-approximation algorithms mentioned above are examples of *greedy* algorithms. They label the points one-by-one in a way that seems best at the time, never undoing earlier choices. In *gradient ascent* one starts from an initial labeling (greedy or random), and then keeps re-positioning sets of one or a few labels as long as this improves the quality of the labeling as a whole. If we imagine the set of all possible labelings as a landscape where height above sea level corresponds to the quality of the labeling, then this method basically tries to ascend to the top of the hill we currently find ourselves on. When the initial random labeling puts us near a small hill, we may get stuck on it while large mountains loom on the horizon. *Simulated annealing* avoids this by sometimes making random changes to the labeling that do not improve it, in an attempt to jump from our hilltop onto the side of a mountain. As computation time increases, the chance and severity of random changes is scaled down, going from completely random perturbations at the start to following the gradient ascent method at the end. Simulated annealing was first applied to labeling by Christensen et al. [20]. Its high solution quality has made it the yardstick for all labeling heuristics since to measure themselves against.

Some other techniques to avoid small hills in the solution landscape include tabu search, evolutionary algorithms, and ant colony optimization. In *tabu search* one keeps track of recently encountered solutions and marks them as "tabu" (forbidden), forcing the search to proceed downhill and (hopefully) towards a mountain. This idea was applied to labeling by Yamamoto et al. [95], and led them to even better labelings than obtained through simulated annealing. *Evolutionary algorithms* form a class of techniques inspired by biological evolution. Instead of trying to improve upon a single solution (an *individual*), they keep track of multiple solutions (the *population*). At each step the properties of the best solutions (*fittest individuals*) are combined to form new solutions (*offspring*), while the worst solutions are removed (*natural selection*).

While some individuals may get stuck on hills, others should reach the mountains if the population is large enough. This has been applied to labeling by many [24, 81, 91, 23], also leading to improvement over simulated annealing. Another technique inspired by nature is *ant colony optimization*. Here the solution space is modeled by a graph where paths represent solutions. The population then consists of ants which wander through this graph randomly. As they go, the ants deposit *pheromones* along the edges they visit. The more pheromones deposited on an edge, the more likely the ants are to visit it in their wandering. Over time, pheromones evaporate to avoid getting stuck with a suboptimal solution. Schreyer and Raidl [87] applied these techniques to map labeling. They found they could produce solutions with comparable quality to those obtained with simulated annealing, but not significantly better as in the case of tabu search and evolutionary algorithms.

For the fixed-position models, the optimal labeling can be described by an *integer linear program*. Each label candidate is modeled by a variable whose value can be 0 or 1, denoting whether that label candidate is picked or not. Linear constraints then specify that each point may have at most one of its label candidates picked, and that any two overlapping label candidates are not both picked. A feasible solution with a maximum number of picked labels then yields an optimal labeling. As integer linear programming is itself NP-hard [84], this does not help directly. A tremendous body of work has gone into both heuristic and exact methods for (integer) linear programming, however, as many important optimization problems from diverse fields fit in its framework. Zoraster [98, 99] was the first to exploit this for labeling.

The *force-based* approach to labeling was first proposed by Hirsch [39]. Starting from a random labeling in a 1- or 4-slider model, an imaginary force is applied to every pair of overlapping labels, based on the area of overlap. These forces push overlapping labels apart, resulting in new label positions. This process is repeated until an acceptable labeling results. As in gradient descent, this may result in poor solutions if we were unlucky in choosing our initial random labeling. Stadler et al. [88] propose to instead start from a more intelligently chosen initial labeling, which they compute using image processing techniques. Ebner et al. [27, 28] instead combine the force-based approach with simulated annealing.

1.2.3 Dynamic point labeling

A natural generalization of static point labeling is dynamic point labeling. Here the point set changes over time, by points being added and removed, and/or by points moving continuously. This can be inherent to the point set (as in air traffic control), or be the result of the user panning, rotating, and zooming their view of the point set (as in interactive mapping applications). For such a *dynamic point set* P we then seek a *dynamic labeling* $\mathcal{L}$, which for any point in time t assigns a static labeling $\mathcal{L}(t)$ to the set $P(t)$ of points present at

time t. To ensure readability in a static labeling it is enough to have non-overlapping labels of sufficient size. In a dynamic setting, text that "jumps around" significantly impairs reading comprehension, while text scrolling slowly and smoothly does not [47]. As such, we desire that a dynamic labeling changes slowly and continuously as the point set changes. Our eyes can then track the labels as we read them, and will not be drawn to discontinuities in the labeling where none existed in the point set.

Generalizing static labeling problems. Each of the algorithmic labeling problems on static points mentioned in the previous section can be generalized to dynamic point sets. In the case of the decision problem, the question becomes whether there exists a dynamic labeling $\mathcal{L}$ for a given dynamic point set P, in which all points are labeled without overlap during the entirety of a given time interval $[a, b]$.

For the optimization problems, there are several reasonable generalizations. Let f be a function that measures the quality of a static labeling (the size of its labels, or the total number of labels). We then seek a dynamic labeling $\mathcal{L}$ that maximizes $\min\{f(\mathcal{L}(t)) \mid a \leqslant t \leqslant b\}$, for example, or that maximizes $\int_a^b f(\mathcal{L}(t))\,dt$. For size maximization this has not been done previously, nor does it seem desirable. Continuously scaling labels would be akin to someone moving the book we are trying to read back and forth: not a pleasant reading experience at all. For number maximization such a generalizations would result in labels appearing and disappearing, even when no new points appear and no existing points disappear. Such (dis)appearances are inherently discrete events that can disturb the user. They can be alleviated somewhat by making the labels fade in and out gradually, but it would still be prudent to limit such occurrences. Hence, other optimization criteria (such as the free-label maximization problem introduced below) are required.

Labeling interactive maps. The simplest instances of dynamic point labeling concern a static point set that is viewed through a rectangular *viewport* that is continuously panned, rotated, or zoomed by the user. Such interactive mapping applications are common on desktop computers, smartphones, and car navigation systems alike. It is then usually desired that the individual static labelings making up the dynamic labeling are not a function of time, but of the viewport. In other words, if we pan, rotate, and/or zoom the viewport, and then return it to its previous state, the displayed static labeling should be the same as before.

In order to achieve interactive labeling speeds, early efforts separated this task into two parts [77, 76, 78]. First, an expensive preprocessing phase builds a data structure on the points and label candidates. During interaction this data structure is then queried with the panning and zooming viewport. This produces the subset of the points and label candidates which are in the view, along with additional information that allows them to be labeled quickly. Later efforts achieve interactive labeling speeds without preprocessing, through dif-

ferent algorithmic techniques [67, 56, 21], or the use of the massively-parallel graphics-processing hardware on modern computers [89, 90].

Most of these methods make no effort to ensure that the dynamic labeling changes smoothly over time, focusing only on the quality of its individual static labelings and the speed with which they are computed. Poon and Shin [78] attempt to remedy this by precomputing labelings at a discrete set of zoom levels, and then interpolating between the nearest two for a given zoom level. Vaaraniemi et al. [90] generalize Hirsch's force-based method to dynamic labeling, and remove not just overlapping labels but also fast-moving labels.

Been et al. [7, 8] were the first to rigorously enforce a smooth labeling, as well as the first to obtain theoretical guarantees on the quality of the dynamic labeling. They allow a label to only be visible on a single, contiguous range of zoom levels. Their aim is then to maximize the summed length of these ranges (so $\int_a^b f(\mathcal{L}(t))\,dt$ in the formulation above), under the condition that no labels may overlap at any zoom level. They assume the positions of all (unit-square) labels are given, and then achieve constant-factor approximation algorithms for determining the zoom ranges at which they should be displayed. In a similar setting with continuous rotation instead of zooming, Gemsa et al. [34] achieve a PTAS for unit-square labels. Since both of these results presume the label positions as given, computing a good placement for them in this setting remains an open problem. As a small first step in this direction, Gemsa et al. [35] give an FPTAS for continuous zooming of sliding labels in one dimension (that is, all points are on a horizontal line and all labels are horizontal segments).

Labeling moving points. The general dynamic-labeling problem concerns a set of points that move continuously, and that may appear or disappear even without reference to any finite viewport. Air traffic controllers, for example, monitor airplanes in a box-like region of the skies as moving points on their screens. Points are removed not just at the edges of the screen, but also when airplanes rise or descend to altitudes outside the box. Another application area is virtual and augmented reality, where the user walks through a (real or simulated) 3-dimensional world that is annotated with labels. This causes objects to move on the 2-dimensional display, and can cause closer objects to obscure farther objects from the view. We then want to label only the visible objects.

Some of the above heuristics for number maximization in interactive maps can also be applied to this more general setting (namely those that work without preprocessing). In addition, there are number-maximization heuristics specifically for augmented and virtual reality [9, 82, 83]. Number maximization may not be in the user's best interest, however. Allendoerfer et al. [3] found that air traffic controllers were more hindered in their jobs by a fast label movement to prevent overlap, than they were by labels actually overlapping. Peterson et al. [71, 73] found that in virtual reality, label movement is more likely to go unnoticed for overlapping labels than for non-overlapping labels. Moreover,

they found that using *stereoscopic displays* the negative effects of on-screen overlap of labels can be greatly reduced, by placing overlapping labels at different depths [74, 75]. In such displays, having the association between labels and points clear at all times was found to be a more important criterion in dynamic label placement than avoiding label overlap [72].

These studies suggest that for labeling moving points one should perhaps not use the traditional number-maximization formulation of the labeling problem. The focus should be first and foremost on labels being clearly associated with their points and moving slowly, and only then on avoiding overlap. To this end we introduce a new variant of the labeling problem which we call *free-label maximization*. It requires that all points are labeled at all times (as in size maximization) with labels of a given size (as in number maximization), matching European air-traffic safety regulations [25]. We then aim to maximize the number of *free* labels, that is, labels which do not overlap other labels. For the most part, air-traffic controllers have had to perform this task by hand, on top of their actual job of monitoring air-traffic for potential collisions. There are some heuristic algorithms for this task, such as Duverger's [26] which labels points moving in discrete steps by a greedy method. Such methods have not been widely adopted, however, possibly because early heuristic algorithms were found to not be very effective [25].

1.2.4 Our results

For static point labeling, both heuristics and approximation algorithms abound. Dynamic point labeling has seen far less success, as discussed above. In particular, no algorithms with proved approximation ratios exist, except for the relatively simple special cases of zooming or rotating a static point set to be labeled in the 1-position model. We believe this relative lack of results for dynamic point labeling is not due to a lack of attempts, but because dynamic point labeling should be much harder than its static counterpart. In Chapter 3 we prove and quantify this intuition. Specifically, we consider the decision problem of whether there exists a dynamic labeling for a given dynamic point set, in which all points are labeled without overlap during the entirety of a given time interval. We show that this problem is strongly PSPACE-complete for the 4-position, 2-slider, and 4-slider models, even with unit-square labels. This immediately implies that any desired generalization to dynamic point sets of size maximization, number maximization, or free-label maximization is strongly PSPACE-hard. These results apply even if the dynamic nature of the point set is drastically restricted. For size maximization we can even show that no polynomial-time $(2-\varepsilon)$-approximation algorithm can exist for any $\varepsilon > 0$, except in the unlikely event that P = PSPACE. An extended abstract of these results was presented at the *29th European Workshop on Computational Geometry* [16].

Of these PSPACE-hard dynamic labeling problems, free-label maximization is in especially dire need of good algorithms. As a first step in this direc-

tion, Chapter 4 studies free-label maximization for a set of n static points. We show that each of the fixed-position and slider models admits a PTAS and an $O(n \log n)$-time constant-factor approximation algorithm for unit-square labels. The approximation ratio of the latter algorithm is 6 for the 1-slider model, 7 for the 2-position model, 22 for the 4-position and 2-slider models, and 32 for the 4-slider model. These results were published in the journal *Computational Geometry: Theory and Applications* [11]. Apart from having merit in their own right, we shall use these static labeling algorithms in Chapter 5 as subroutines in a heuristic dynamic labeling. We were unfortunately not able to prove a guaranteed approximation ratio for this heuristic. We have, however, evaluated its performance in experiments, showing that the algorithm produces good dynamic labelings in practice. By varying two parameters to the algorithm, a trade-off is achieved between how many labels are free for how long, and how fast the labels are moving. Even modest sacrifices in label freeness greatly reduce the speeds of the labels. These results have been accepted to be presented at the *21st European Symposium on Algorithms* [13].

Computing Push Plans for Disk-Shaped Robots

2

Suppose we want to move an object that is too large or too heavy to carry. For example, say we wish to move a big, heavy potted plant across an office, and suppose we have a more or less cylindrical robot at our disposal (an automated vacuum cleaner or floor polisher, say). When viewed from above, this is essentially a 2-dimensional problem, involving

- a disk-shaped *pusher* P of radius r_p (the robot),
- a disk-shaped *object* O of radius r_o (the pot), and
- a set of n non-intersecting line-segment *obstacles* Γ (the outlines of the office furniture).

For the pusher to achieve our goal, it needs to compute a *push plan*, a path avoiding the obstacles that will result in it pushing the object to its destination.

As explained in Section 1.1.3, the state-of-the-art algorithm for this problem is due to Nieuwenhuisen et al. [69]. Their algorithm is an adaptation of the *Rapidly-exploring Random Trees (RRT)* path-planning algorithm of LaValle and Kuffner [54]. It incrementally builds up a tree of obstacle-avoiding pushing motions with the initial object position at the root and eventually having the object's destination as a leaf. To construct the edges of the tree, the algorithm uses a subroutine that solves a slightly simpler problem: given an obstacle-avoiding path τ for O (rather than just its destination), compute an obstacle-avoiding push plan σ for P which makes it push O along τ as far as possible. Much of the efficiency and effectiveness of the algorithm depends on the method used to solve this subproblem.

In this chapter, we greatly improve on how Nieuwenhuisen et al. solve the above subproblem. Specifically, our method does not need $O(n^2 \log n)$ time and $O(n^2)$ space for preprocessing the obstacles, and can handle more general classes of object paths. Moreover, if there exists a *contact-preserving push plan*,

in which the pusher maintains contact with the object at all times, then we compute the shortest such push plan. As we will show in Section 2.4, however, sometimes the pusher must let go of the object on occasion to resume pushing from a different location. We can also compute such *unrestricted push plans*, although we can then not guarantee finding the shortest one.

2.1 The configuration space

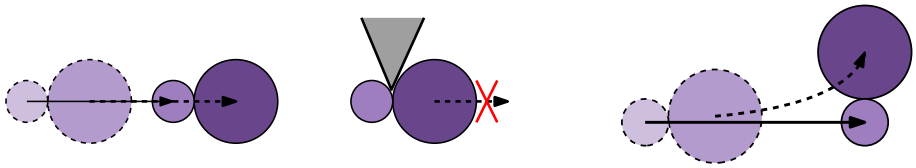
A general-purpose technique for path planning is to translate the problem from the *work space* into the *configuration space* [51]. The work space is the environment in which the robot has to find a path. A *configuration* is one specific placement of the robot in this space, specified by f parameters, where f is the number of degrees of freedom of the robot. Each point in the f-dimensional configuration space corresponds to a configuration in the work space. Some configurations are invalid because the robot would intersect an obstacle and these form the *forbidden (configuration) space*. The remainder is the *free (configuration) space*, and a path through it represents a solution to the original path-planning problem in the work space.

To apply this technique to our problem, we first clarify a few details left out from the problem statement, and then discuss what our configuration space looks like and how to compute it. Finding paths through the configuration space, and mapping these back to push plans, is discussed in Sections 2.2 and 2.3.

2.1.1 Preliminaries

We assume (as does the prior work of Nieuwenhuisen et al. [69] and Agarwal et al. [2]) that pushing is *quasi-static* [70]. That is, when pushing stops the object also stops instantly. This is realistic if pushing is done very slowly, or if there is a large amount of friction between the disks and the floor. To push the object along a straight-line path in the absence of obstacles, the ray from the pusher's center to the object's center will have to line up *exactly* with the desired direction of motion, as in Figure 2.1(a). When pushing even slightly off-center, the object's motion will instead be curved. If we assume the point of contact between the two disks never "slips" during a push, the result is a so-called *hockey-stick curve* [2], as seen in Figure 2.1(b). Given such a straight-line or hockey-stick motion for the object, the pusher motion that will accomplish it is fixed.

In contrast to such *non-compliant motions*, a *compliant motion* uses obstacles as a guide for the object, as in Figure 2.2. On the left the object is pushed straight along the length of an obstacle, in the middle the object is pushed in a circular arc around an obstacle endpoint. Compliant motions are more robust in the presence of sensor inaccuracies, and allow the pusher to avoid obstacles more easily, as the same object motion can be achieved by a whole range of



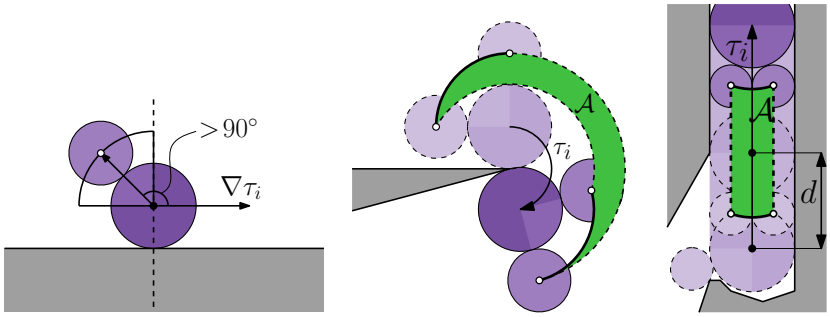
(a) (b)

Figure 2.1: (a) A straight-line non-compliant path section, and (b) a hockey-stick non-compliant path section. In all figures in this chapter, the pusher P is depicted as the smaller, lighter disk and the object O as the larger, darker disk. Obstacles are shown as fat, black lines and the areas they bound are filled with gray. Motion is depicted by arrows; the boundaries of P and O at the current position are solid, and are dashed at past or future positions.

pusher motions. At any point in time, there is a circular arc of allowed positions for the pusher's center from which pushing would result in the desired object motion. We call this arc the *push range*; it can be computed from the friction coefficients between the pusher and the object, and between the object and the obstacle [68].

We assume that the given object path $\tau : [0, 1] \to \mathbb{R}^2$ consists of k constant-complexity curves $\tau_1, \ldots, \tau_k$ connected end-to-end. By a *constant-complexity curve* we mean that such a curve takes $O(1)$ space to represent, and that we can perform several basic operations on it in $O(1)$ time: computing tangents to a curve through a given point, and computing bi-tangents and points of intersection between two curves. We call each such piece of τ a *path section*. A path section represents one of the four aforementioned motions: a straight-line or hockey-stick non-compliant motion, or a straight-line or circular compliant motion. To avoid four-way case distinctions throughout this chapter, we introduce an abstract *well-behaved* path section which generalizes these four. As an added benefit, this makes our algorithm more general: we can easily handle other types of path sections, as long as they are well-behaved. For example, one may wish to incorporate non-compliant motions for which the pusher has to follow a non-straight path, or in which the contact between the two disks *does* slip during the push. We call a path section τ_i well-behaved if it is a convex, constant-complexity curve and satisfies the following conditions (see Figures 2.2 and 2.3):

(W1) We can compute the push range $\mathrm{pr}_i(s)$ for any object position $\tau_i(s)$ along τ_i in $O(1)$ time. Furthermore, this push range is such that the ray from O's center in the direction of τ_i always forms an angle of more than 90° with a ray from O's center to a pushing position. (This is natural, since otherwise the pusher would pull the object rather than push it.)



(W1) (W2) (W3)

Figure 2.2: The three properties satisfied by well-behaved path sections. The push range (a circular arc of valid pusher positions) is shown at a few object locations, further accentuated by a sector on the object.

(W2) Let $\mathcal{A} = \bigcup_{s\in[0,1]} \mathrm{pr}_i(s)$ be the area swept out by the push range as the object moves along τ_i. Then $\mathcal{A}$ does not intersect itself, and $\mathcal{A}$ is bounded by four convex, constant-complexity curves (the push ranges at either end of τ_i, and the paths traced out by the two end points of the push range) which can be computed in $O(1)$ time.

(W3) For compliant sections there is a constant $d = O(r_o)$ such that, after the object has moved a distance d along the path section, the push range becomes such that the pusher can remain in the sweep area of the object for the rest of the section. Furthermore, the smaller push range that keeps the pusher in the object's sweep area satisfies (W1) and (W2).

For correctness of our algorithms, (W1) and (W2) suffice, but (W3) allows us to derive better running times in case the obstacles are not too densely packed, as will be discussed in Section 2.1.4. Condition (W3) basically states that for long path sections we need only maneuver around obstacles during the beginning of the path section. In compliant sections the pusher can avoid any later obstacles by "hiding" behind the object, and in non-compliant sections obstacles cannot be avoided at all (see Figure 2.1(a)). Note that circular compliant sections satisfy (W3) trivially as their total length is $O(r_o)$, and straight-line compliant path sections can be shown to satisfy (W3) as well [68], as seen informally on the right in Figure 2.2.

It is then fairly straightforward to see that all three criteria are satisfied by straight-line and circular compliant path sections, as well as by straight-line non-compliant path sections. As these are the path sections handled by the algorithm of Nieuwenhuisen et al. [69], our algorithm is at least as general as theirs. A hockey-stick curve (whose equation is given in Section 2.4) is not strictly of constant complexity by our definition, but it can be approximated

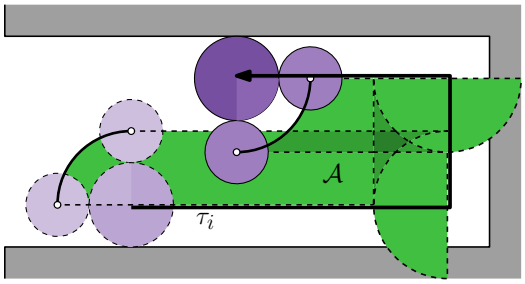


Figure 2.3: A path section where well-behavedness property (W2) is *not* satisfied (the area swept out by the push range self-intersects). The path section *can* be broken up into three well-behaved path sections.

to any desired degree by a sequence of quadratic Bézier curves. Thus hockey-stick non-compliant path sections can also be seen as well-behaved, and we can handle them uniformly with other types of path sections. In Nieuwenhuisen's approach they are handled outside of the main algorithm by ad hoc numerical methods [69].

2.1.2 Shape of the configuration space

A configuration in our problem is a placement of both the pusher and the object in the work space. Recall that the object is restricted to move along a given path τ, and assume for now that the pusher and object maintain contact at all times (this latter restriction will be lifted in Section 2.3). Under these conditions, the configuration space is 2-dimensional. The point $(s, \theta) \in [0, 1] \times \mathcal{S}^1$ in the configuration space will represent the configuration with the object's center at $\tau(s)$ and with θ being the *pushing angle*: the angle that the ray from the pusher's center to the object's center makes with the positive x-axis. (Note that the configuration space is cylindrical, but for clarity we will depict it "unfolded" as a rectangle.)

We assume that path τ does not take the object through any obstacles. A configuration can then only be invalid for two reasons: either the pusher intersects an obstacle, or the pusher is outside of the push range. We therefore consider the forbidden space to be the union of two kinds of shapes. A *configuration-space obstacle* $\mathcal{C}_\gamma$ consists of the configurations where the pusher intersects obstacle γ. A *forbidden push range* FPR_i consists of the configurations where the object is on the interior of path section τ_i and the pusher is outside the push range. By $\mathcal{C}_{\gamma,i}$ we will mean the restriction of $\mathcal{C}_\gamma$ to configurations with the object on path section τ_i. The forbidden space is then the union of $k(n + 1)$ shapes: n obstacles and one forbidden push range for each of the k path sections. Figure 2.4(a)–(b) shows an example for one path section.

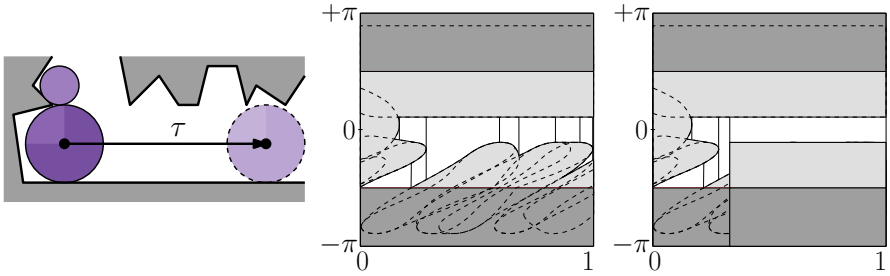


Figure 2.4: (a) An example work space, (b) its configuration space, and (c) its reduced configuration space (defined below). The s-axis is horizontal, the θ-axis is vertical. Configuration-space obstacles are drawn dashed in light gray, the forbidden push range is drawn in dark gray.

Theorem 2.1. *The configuration space for each path section has complexity $O(n)$ (that is, the boundary of the forbidden space consists of $O(n)$ vertices and constant-complexity curves between them), and thus the total configuration space has complexity $O(kn)$.*

Proof. Since a path section τ_i has constant complexity, so do FPR_i and $\mathcal{C}_{\gamma,i}$ for all obstacles $\gamma \in \Gamma$. We will prove that $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. It then follows that $\mathrm{FPR}_i \cup \bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$, the forbidden space for one path section, also has complexity $O(n)$, yielding $O(kn)$ in total.

The boundary of $\mathcal{C}_\gamma$ corresponds to configurations where the pusher is compliant with γ. Such pusher positions all lie at distance r_p from γ and thus form the boundary of the "capsule" $\gamma \oplus D(r_p)$. Here "$\oplus$" denotes the Minkowski sum ($A \oplus B = \{\, x + y \mid x \in A,\, y \in B \,\}$), and $D(r)$ denotes an origin-centered disk of radius r. A point of intersection of $\mathcal{C}_{\gamma_1,i}$ and $\mathcal{C}_{\gamma_2,i}$ corresponds to a configuration where the pusher is compliant with both γ_1 and γ_2, and that pusher position must thus be an intersection of the corresponding capsules. These capsules form a collection of *pseudodisks* [48], and therefore have a union complexity of $O(n)$. Thus there can only be $O(n)$ positions where the pusher would be compliant with more than one obstacle. Each of these pusher positions could show up in the configuration space more than once, since path τ_i could take the object past this point multiple times. This cannot happen more than $O(1)$ times, however, since τ_i is a convex, constant-complexity curve. Thus $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. □

2.1.3 Computing the configuration space

To compute the configuration space in a form that allows us to easily compute a push plan, we do the following.

1. Compute $\mathcal{C}_{\gamma,i}$ for all $\gamma \in \Gamma$ and $\tau_i \in \tau$.
2. Compute FPR_i for all $\tau_i \in \tau$.
3. Take the union of these shapes to get the forbidden space.
4. Divide the free space into *cells* by a vertical decomposition.
5. Create the *cell graph* of the decomposition. Nodes in this graph correspond to cells and there is a directed edge from cell c_1 to cell c_2 if and only if c_1's right boundary touches c_2's left boundary.

The running time of this approach is expressed by the following theorem.

Theorem 2.2. *The configuration space can be computed in $O(kn \log^2 n)$ time worst case, or $O(kn \log n)$ expected time, both using $O(kn)$ space.*

Proof. For each of the k path sections, Steps 1 and 2 can be performed in $O(n)$ time as each of these $n+1$ shapes has $O(1)$ complexity. The curves bounding these shapes have complex equations not amenable to the exact intersection computations needed for Step 3. However, we can work with work-space analogues of these shapes to compute their vertices and combinatorial structure. For example, the points of vertical tangency of $\mathcal{C}_{\gamma,i}$ correspond to intersections of τ_i with the capsule $\gamma \oplus D(2r_o + r_p)$, and the intersection of $\mathcal{C}_{\gamma,i}$ with the vertical line $s = s'$ corresponds to the intersection of $\gamma \oplus D(r_p)$ with a circle of radius $2r_o + r_p$ centered at $\tau_i(s')$. Thus, all basic operations required (such as deciding whether one point of vertical tangency lies to the left or to the right of another) can be performed in $O(1)$ time.

With this "trick", Step 3 can then be performed by a deterministic algorithm by Kedem et al. [48] that uses $O(n \log^2 n)$ time, or a randomized incremental algorithm by Miller and Sharir [65] that uses $O(n \log n)$ expected time, both using $O(n)$ space. For Steps 4 and 5 we extend vertical lines from each of the $O(n)$ vertices (including points of vertical tangency) of the forbidden space. This yields a vertical decomposition of the free space into cells. These cells can be computed and connected together with a sweep-line algorithm in $O(n \log n)$ time and $O(n)$ space. □

2.1.4 Low obstacle density

The asymptotic upper bounds derived for our algorithm so far assume nothing about how densely packed the obstacles are. An environment Γ is said to have an obstacle density of λ if λ is the smallest positive number for which any disk D intersects at most λ obstacles $\gamma \in \Gamma$ with $\mathrm{length}(\gamma) \geq \mathrm{diam}(D)$. By property (W3) of well-behaved compliant path sections there is a constant $d = O(r_o)$ such that, after the object has been pushed a distance d along a path section, the pusher can then remain in the (obstacle-free) sweep area of the object for

the rest of the section. The combined sweep area of the object and pusher for this length-d "prefix" of the path section fits in a disk of diameter $d + 2r_o + 4r_p = O(r_o)$, and can thus intersect at most $O(\lambda \cdot {r_o}^2/\delta^2)$ obstacles, where δ is the length of the shortest obstacle. Assuming constant λ and $\delta = \Omega(r_o)$, the configuration space for this prefix has constant complexity.

The complexity of the remaining "suffix" of the path section can be $\Omega(n)$, though, so the total complexity of the configuration space can still be $\Omega(kn)$. However, for this suffix we can replace the $O(n)$-complexity shape $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ by the $O(1)$-complexity shape that forces the pusher to remain in the object's sweep area. This yields the *reduced configuration space* (see Figure 2.4(c)), which admits a push plan if and only if the original configuration space does, but has lower complexity and can be computed more quickly.

Theorem 2.3. *Assuming constant λ and $\delta = \Omega(r_o)$, the reduced configuration space has complexity $O(1)$ per path section (that is, $O(k)$ in total), and can be computed in $O((k+n)\log(k+n))$ time using $O(k+n)$ space.*

Proof. For the reduced configuration space, computing $\mathcal{C}_{\gamma,i}$ individually for each $\gamma \in \Gamma$ and $\tau_i \in \tau$ is wasteful. Most of these kn shapes will either be empty (if the distance between τ_i and γ is too great), or irrelevant (if τ_i is compliant and only comes close to γ past its length-d prefix). Let τ_i' denote the length-d prefix of τ_i if τ_i is a compliant section, and $\tau_i' = \tau_i$ otherwise. Then $\mathcal{C}_{\gamma,i}$ is relevant and non-empty if and only if the curve τ_i' intersects the capsule $\gamma \oplus D(2r_o + r_p)$. We use an algorithm due to Balaban [5] to find the I points of intersections between these k curves and n capsules in $O((k+n)\log(k+n) + I)$ time and $O(k+n)$ space. From the above discussion it follows that each compliant path section contributes only $O(1)$ to I for the reduced configuration space. If non-compliant sections contribute any intersections, then no push plan can exist and we can abort the intersection computation. Thus, we can determine the relevant obstacles for all path sections in $O((k+n)\log(k+n))$ time. Since each section only has $O(1)$ relevant obstacles, the remaining steps to compute the reduced configuration space can be done in $O(1)$ time per path section. □

2.2 Pushing while maintaining contact

Not every path through the free space actually yields a push plan. The path through the configuration space needs to be *s-monotone*, that is, any "vertical line" (having a constant value for s) must intersect the path in at most one point. If a path through the configuration space is not s-monotone, then the object would be going backwards on occasion, for which the pusher would have to pull. To prevent this, we remove from the cell graph all cells that are not reachable from the starting configuration by a valid contact-preserving push plan. It is then fairly simple to find an arbitrary s-monotone path in linear time, by following cell boundaries (which are s-monotone). Such a path can

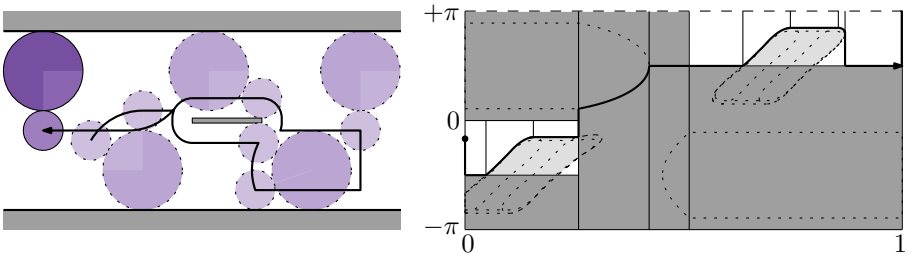


Figure 2.5: Following the boundaries of the configuration-space cells may yield a very sub-optimal push plan.

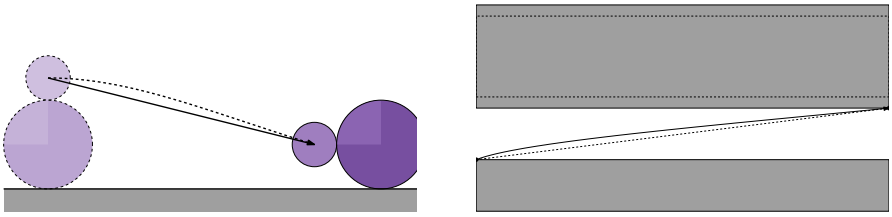


Figure 2.6: The dotted push plan is the Euclidean shortest path in the configuration space, but not in the work space. The solid push plan is the Euclidean shortest path in the work space, but not in the configuration space.

lead to a lot of unnecessary motion for the pusher, however (see Figure 2.5). It is tempting to instead compute a Euclidean shortest path through the reachable cells. While this would also yield an s-monotone path, it does not necessarily minimize the pusher's movement in the work space either (see Figure 2.6). We can circumvent this problem by performing our computations in the work space, instead of in the configuration space.

2.2.1 Cells in the work space

Each cell of the free-space decomposition corresponds to a contiguous subset of the valid configurations. The pusher positions of these configurations also form a contiguous region in the work space. We call this region the corresponding *work-space cell*. All work-space cells glued together by their common boundaries form the region that P may move in to accomplish O's desired motion, provided we consider non-adjacent cells to be on a different "layer". We cannot just take the union of the cells, as P could then take a shortcut and lose O along the way. Figure 2.3 showed such a situation: moving through the union of the work-space cells, P could push O to the lower-right corner, then leave it there and move on towards the top left. This way P always stays within the union of the work-space cells, but still fails to push O to its destination. If P instead

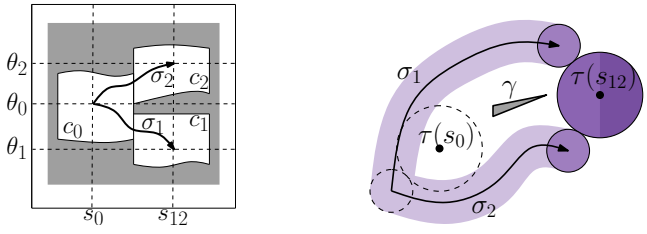


Figure 2.7: Hypothetical situation in which a configuration-space cell would have an out-degree greater than one.

traverses the work-space cells in sequence along a path in the cell graph, then such a situation cannot occur. As the following lemma shows, this cell sequence is unique.

Lemma 2.1. *All cells in the cell graph have an out-degree of at most one.*

Proof. Suppose cell c_0 has out-edges to cells c_1 and c_2. From any $(s_0, \theta_0) \in c_0$ there must then be a push plan σ_1 to any $(s_{12}, \theta_1) \in c_1$ and a push plan σ_2 to any $(s_{12}, \theta_2) \in c_2$, as in Figure 2.7(a). Property (W1) of well-behaved path sections then implies that any obstacle γ that causes c_1 and c_2 to be separate cells must be in the region between the areas swept out by the pusher along σ_1 and σ_2, and the area occupied by the object positioned at $\tau(s_{12})$, as in Figure 2.7(b). But because the pusher maintains contact with the object at all times, this region must lie entirely within the area swept out by the object, which we assumed was obstacle free. □

To compute a shortest contact-preserving push plan, we need only compute a shortest path through this sequence of work-space cells. When used as a subroutine in the RRT-based algorithm mentioned in the introduction to this chapter, we typically seek the shortest path from a point (the initial pusher position) to a circular arc (the push range at the goal position of the object), with both the point and the arc lying on the boundaries of work-space cells. Perhaps, however, we are free to choose the initial pusher position, in which case we also need to be able to compute the shortest path between two circular arcs (the push ranges at the initial and final positions of the object). Lastly, if the pusher's final position is specified, then we need to compute a shortest path from a point or arc to a point. To simplify the discussion, we will first describe a linear-time method to compute shortest point-to-point, point-to-edge, and edge-to-edge paths in a chain of triangles. We then generalize this approach to point-to-point, point-to-arc, and arc-to-arc paths in our chain of curved work-space cells.

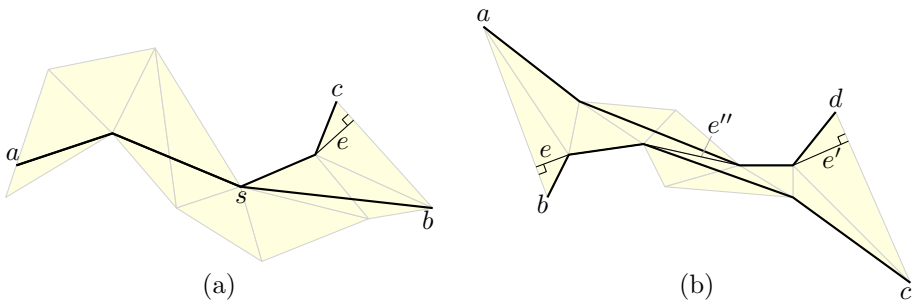


Figure 2.8: (a) A funnel, used to compute a shortest point-to-edge path $\pi(a, bc)$. (b) An hourglass, used to compute a shortest edge-to-edge path $\pi(ab, cd)$.

For chains of triangles (and triangulated simple polygons in general), Lee and Preparata [55] gave a linear-time algorithm to compute a shortest point-to-point path. Their algorithm processes the triangles one-by-one in sequence, and will still work if later triangles overlap earlier triangles. In this case the output will be the (possibly self-intersecting) shortest path that traverses the triangles in sequence, but may not necessarily be the shortest path through the union of the triangles. This is exactly what we need.

To compute a shortest path $\pi(a, bc)$ from a point a to an edge bc, consider the two shortest paths $\pi(a, b)$ and $\pi(a, c)$ from a to the edge's endpoints. These paths will be identical up to some point s (possibly equal to a), and then diverge into a *funnel* [55] formed by the subpaths $\pi(s, b)$ and $\pi(s, c)$. Figure 2.8(a) shows an example. The desired path $\pi(a, bc)$ must start with $\pi(a, s)$, end with a (possibly degenerate) edge e tangent to the funnel and perpendicular to bc, and follow the funnel in between the two. Here "tangent to the funnel" is to be understood as tangent to one of its subpaths, and e may indeed connect to s directly. The algorithm of Lee and Preparata yields this funnel for free as a by-product, and finding the proper tangent e may be done by brute force as the funnel has linear size.

To compute a shortest path $\pi(ab, cd)$ from edge ab to edge cd, with (a, b, c, d) being the counter-clockwise order in which these vertices appear on the boundary of R, we consider the *hourglass* [37] formed by the union of $\pi(a, d)$ and $\pi(b, c)$. Figure 2.8(b) shows an example. The desired path $\pi(ab, cd)$ must start with a (possibly degenerate) edge e tangent to the hourglass and perpendicular to ab. Likewise, the path must end with a (possibly degenerate) edge e' (possibly equal to e) that is tangent to the hourglass and perpendicular to cd. From e onward, and from e' backward, the path follows the edges of the hourglass. If these two chains meet, then that completes the path $\pi(ab, cd)$. Otherwise, the two chains are connected by an edge e'' tangent to both. As for funnels, "tangent to the hourglass" is to be understood as tangent to one of these two chains. The hourglass may be computed by two runs of the algorithm of Lee

and Preparata, and this yields the needed bi-tangent e'' as a by-product [37]. Finding the proper starting and ending tangents may again be done by brute force.

The above machinery of funnels and hourglasses for triangulated simple polygons has been generalized to *pseudo-triangulated* simple *splinegons* by García-López and Ramos [33]. A splinegon S with *underlying polygon* P is the result of replacing the edges of P by constant-complexity convex curves. A pseudo-triangle is a 3-vertex splinegon where all three boundary curves bend inward (in other words, a pseudo-triangle is a subset of its underlying triangle). The needed pseudo-triangulation of a splinegon can be obtained in linear-time, given a triangulation of its underlying polygon [33]. To sum up:

Theorem 2.4. *Given a chain of splinegons of m vertices in total, we can compute in $O(m)$ time and space a shortest path traversing them in sequence. This is the case for shortest point-to-point, point-to-curve, and curve-to-curve paths.*

2.2.2 Computing a shortest contact-preserving push plan

We have now established all the needed machinery to compute shortest contact-preserving push plans. To state the result without having to make a case distinction between high and low obstacle density, we let q denote the maximal complexity of the configuration space for one path section. The unreduced configuration space has $q = O(n)$, while the reduced configuration space under low obstacle density has $q = O(1)$. Note, though, that a shortest push plan through the reduced configuration space is not necessarily as short as one through the unreduced configuration space. Hence $q = O(n)$ if we require a shortest push plan.

Theorem 2.5. *Given a k-section configuration space with complexity $O(q)$ per path section, a shortest contact-preserving push plan (of complexity $O(kq)$) through this space can be computed in $O(kq)$ time and space.*

Proof. A work-space cell w is the area $\mathcal{A}$ swept out by the push range over a piece of the object's path, minus the positions where the pusher would intersect any obstacles. Because of the way configuration-space cells are defined by a vertical decomposition, at most two obstacles can influence the shape of one work-space cell. Thus w is the set difference of $\mathcal{A}$ with at most two capsules. By property (W2) of well-behaved path sections, $\mathcal{A}$ is bounded by a constant number of convex, constant-complexity curves, so w must be as well. Thus a work-space cell is a splinegon with $O(1)$ vertices, which can be pseudo-triangulated in $O(1)$ time [33]. By Lemma 2.1, the reachable configuration space is a linear chain of $O(kq)$ cells. By Thereom 2.4 we may compute a shortest path through this chain in $O(kq)$ time. This path corresponds to a shortest contact-preserving push plan. □

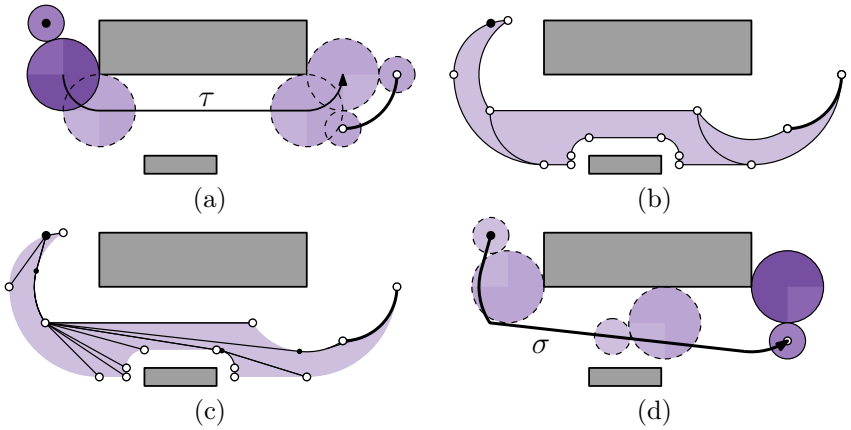


Figure 2.9: (a) An example work space, and (b) its corresponding work-space cells. The pusher's starting point is drawn as a black dot, and its destination curve is drawn bold. (c) The final step in constructing (b)'s shortest-path tree. (d) The resulting (optimal) push plan.

Figure 2.9(a)–(b) shows an example of a work space and its work-space cells. Here the initial pusher position is specified, but the final pusher position is not, so we seek a shortest point-to-curve path. The cases of curve-to-curve and point-to-point would look similar.

2.3 Pushing and releasing

Until now we have assumed the pusher can maintain contact with the object at all times. However, the situation depicted in Figure 2.10(a)–(b) does not admit such contact-preserving push plans. (In fact, this is true for *any* object path with the same start and end point, as we shall prove in Section 2.4.) It does admit an unrestricted push plan, however, as can be seen in Figure 2.10(b)–(c).

2.3.1 Canonical releasing positions

Whenever the push range is split into multiple contiguous ranges by obstacles, it may make sense for P to let go of O and try to reach one of these other positions. In the configuration space this situation corresponds to a vertical line intersecting multiple cells. In general, there are infinitely many such *potential releasing positions*, thus it is infeasible to try them all. Instead, we consider only vertical lines that go through a vertex of a cell or of a configuration-space obstacle (where points of vertical tangency are also considered vertices). We call the resulting set of $O(kq)$ positions the *canonical releasing positions*. As before, $O(q)$ here denotes the complexity of the configuration space of a single path

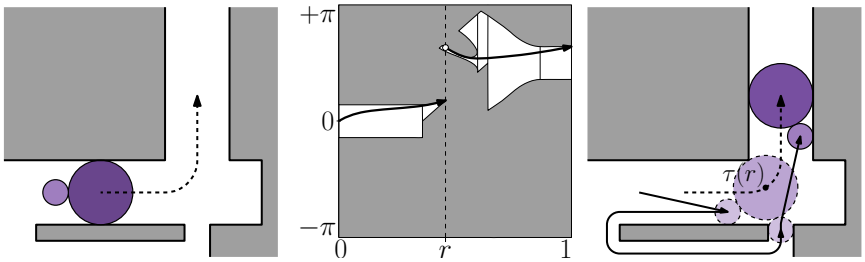


Figure 2.10: (a) An example work space for which no contact-preserving push plan exists, (b) its configuration space, and (c) an unrestricted push plan for it, doing one release at position $\tau(r)$.

section. It suffices to check only these positions, as expressed by the following lemma.

Lemma 2.2. *If an unrestricted push plan exists for a given input, then there is also an unrestricted push plan where all releases happen at canonical releasing positions as defined above.*

Proof. Suppose we have an unrestricted push plan with one or more releases that do not happen at canonical releasing positions. Let $\tau(r)$ be an object position at which such a release happens, and σ be the path that the pusher follows from the position where it releases the object to the position where it recontacts. Because r is not a canonical releasing position there must be a canonical releasing position $r' < r$ with no other canonical releasing positions in between r' and r.

Suppose the releasing point for σ lies in cell c_1 of the free space and the recontact point in cell c_2. Now imagine moving the object backwards along τ from $\tau(r)$ to $\tau(r')$. In doing this we want to adjust our push plan so it remains valid, making it move a little less through cell c_1, a little more through cell c_2, and adjusting path σ accordingly for its new endpoints.

There are two ways in which this could fail: either the interval of valid pusher positions in which one of the endpoints of path σ resides vanishes, as in Figure 2.11(a), or path σ gets cut off between the obstacles and the object, as in Figure 2.11(b). The former can only happen at a vertex of c_1 or c_2, and the latter can only happen at a vertex of some configuration-space obstacle $\mathcal{C}_\gamma$. But such points would then form canonical releasing points between r and r', contradicting our assumption. □

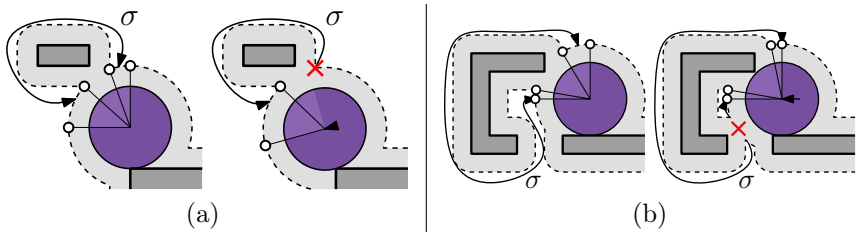


Figure 2.11: The two situations in which moving the object backwards along τ would make us unable to maintain the pusher path σ.

2.3.2 Computing an unrestricted push plan

Restricting ourselves to canonical releasing positions, we cannot guarantee a shortest push plan anymore, so we abandon the work-space-cell approach and instead work with the cell graph directly. The cell graph encodes which configurations are reachable from which other configurations, assuming the pusher and object maintain contact. By adding edges, we will transform this into the *extended cell graph*, which encodes the same connectivity information when the pusher *can* let go of the object. To determine whether an edge between two cells should be added we have to solve a standard path-planning problem for the pusher, with the stationary object being an additional obstacle. The algorithm in more detail is as follows.

1. Compute a *road map* $\mathcal{R}$ for P among the obstacles [51].
2. At each canonical releasing point r:
   a) Add O positioned at $\tau(r)$ as an extra obstacle in $\mathcal{R}$ to get $\mathcal{R}_r$.
   b) Determine the set C_r of cells intersected by a vertical line through r.
   c) Determine for each cell in C_r to which component of $\mathcal{R}_r$ its pusher positions belong.
   d) Add edges in the cell graph between cells sharing a component of $\mathcal{R}_r$.
3. Compute a path through the extended cell graph.
4. Convert the path into a push plan. For edges of the original cell graph this is straightforward. For the extra edges added in Step 2(d) we use the respective roadmap $\mathcal{R}_r$ to find a path for P.

To construct the initial road map (Step 1) we take the union of the capsules $\gamma \oplus D(r_p)$ for $\gamma \in \Gamma$, and then construct a vertical decomposition of its complement. The union can be done in $O(n \log^2 n)$ worst-case time (using an algorithm by Kedem et al. [48]), or in $O(n \log n)$ expected time (using an algorithm by Miller and Sharir [65]). After this preprocessing, the time taken by the remaining steps is expressed by the following theorem.

Theorem 2.6. *Given a road map for P among the obstacles, and a configuration space with complexity $O(q)$ per path section, an unrestricted push plan (of complexity $O(kqn)$) can be computed in $O(kq\log(kq)+kq^2\log n+kqn)$ time using $O(kqn)$ space.*

Proof. While computing the configuration space, all $O(kq)$ canonical releasing positions were already computed so this takes no extra time. For each of them:

- Step 2(a) can be done in $O(n)$ time.
- Step 2(b) can be done using the point location structure associated with the vertical decomposition of the free space, in $O(\log(kq)+|C_r|)$ time with $|C_r| = O(q)$.
- Step 2(c) can be done with the point location structure associated with $\mathcal{R}_r$, in $O(\log n)$ time per cell ($O(q\log n)$ in total).
- Step 2(d) then adds at most $O(q)$ extra edges.

So Step 2 takes $O\big(kq(n+\log(kq)+q\log n\big)$ time, and adds at most $O(kq^2)$ edges to the cell graph. Step 3 can be done by depth-first search in $O(kq^2)$ time. For each of the $O(kq)$ (sequences of) edges in the computed path corresponding to a release-and-recontact maneuver, Step 4 must reconstruct the appropriate $\mathcal{R}_r$ and do a depth-first search in it. Thus Step 4 takes $O(kqn)$ time. □

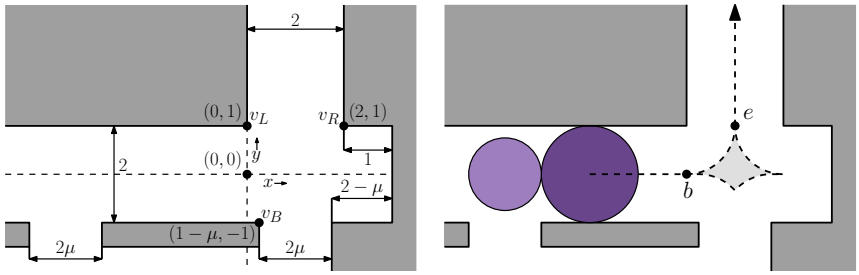
2.4 A proof that releasing can be necessary

In Figure 2.10(a) we saw a work space which permits no contact-preserving push plan for the given object path. One may wonder whether this object path is merely ill-chosen, and whether a different object path *would* admit a contact-preserving push plan. We will prove that this is not the case. We do so under the assumption of a push range of 90°, as depicted in the figures below. In realistic settings the push range will be smaller (due to friction), but that will only strengthen the proof.

Assume, without loss of generality, that $r_o = 1$ and $r_p = \mu$, with $0 < \mu < 1$. Our goal is to get the object from the lower left part of Figure 2.12(a), around the bend, to the upper right part of the figure. The path τ which the object then takes will have to go through the collision-free area depicted in Figure 2.12(b).

Because the horizontal and vertical corridor are exactly as wide as the object, the object can only move through them in one way (and this is easily accomplished by the pusher). We will therefore consider the intermediate section of the object's path connecting these two straight-line sections, that is, the subpath from point b to point e in Figure 2.12(b).

In particular, we will look at the *highest* possible such subpath, that is, the subpath that maximizes the object's distance traveled in the positive y direction



(a) (b)

Figure 2.12: (a) The environment with the coordinate system that we will use. Measurements assume that $r_o = 1$ and $r_p = \mu$. (b) The initial situation, and the area through which the object's center can move without intersecting any obstacles.

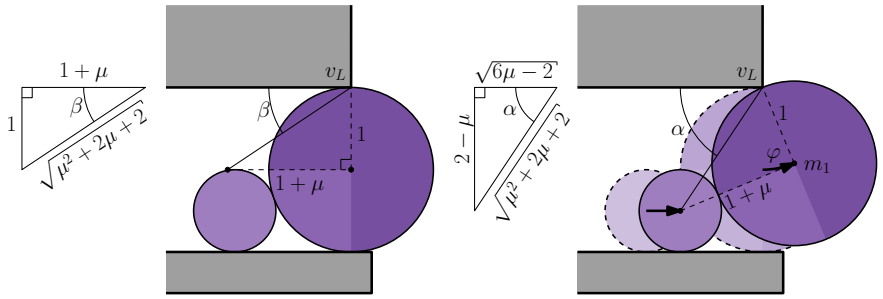
for the distance traveled in the positive x direction. This motion is achieved by minimizing the pusher's y-coordinate at all times. Thus the pusher should start by sliding compliantly along the bottom obstacle. This will turn the object on a circular arc around v_L, but it cannot always reach e via this arc, as stated in the following lemma.

Lemma 2.3. *In the example of Figure 2.12, the pusher cannot push the object compliantly around v_L from b all the way to e if $\mu > 1/3$.*

Proof. In this motion, the object's position describes a circular arc with radius r_o around v_L. To keep the object going along this arc, the ray from the pusher's center to the object's center must intersect the arc, either properly or tangentially. So if there is a position of the object along the arc such that the pusher cannot push tangentially to the arc, the motion cannot be completed. Figure 2.13 shows that this happens when $\sqrt{\mu^2 + 2\mu + 2} > 2 - \mu$. Squaring both sides of the equation and simplifying yields $\mu > 1/3$. □

From now on, assume that $\mu > 1/3$. Lemma 2.3 then implies that if the pusher tries to push the object compliantly around v_L, the object will end up at some point m_1 between b and e. From b to m_1, the object will have turned by some angle $\varphi < \pi/2$ around v_L, where (see Figure 2.13)

$$\begin{aligned} \varphi &= \alpha - \beta \\ \sin(\alpha) &= \frac{2-\mu}{\sqrt{\mu^2+2\mu+2}} \qquad & \cos(\alpha) &= \frac{\sqrt{6\mu-2}}{\sqrt{\mu^2+2\mu+2}} \\ \sin(\beta) &= \frac{1}{\sqrt{\mu^2+2\mu+2}} \qquad & \cos(\beta) &= \frac{1+\mu}{\sqrt{\mu^2+2\mu+2}}. \end{aligned} \tag{2.1}$$



(a) (b)

Figure 2.13: (a) Initial position. (b) Position m_1 where O has turned by an angle $\varphi = a - b$ around v_L and P can no longer turn it further. This happens when $\mu > 1/3$.

Recall that we called the pushing angle θ, and that it is defined as the angle that the line from P's to O's center makes with the positive x-axis. If the pusher continues its straight line along the bottom obstacle, the object will move away from v_L to the right and upwards, thus increasing θ. The resulting motion was studied by Agarwal et al. [2], and they refer to it as a *hockey-stick curve*. As a function of θ, this motion is given by the following equations, where $\varphi \le \theta \le \pi/2$.

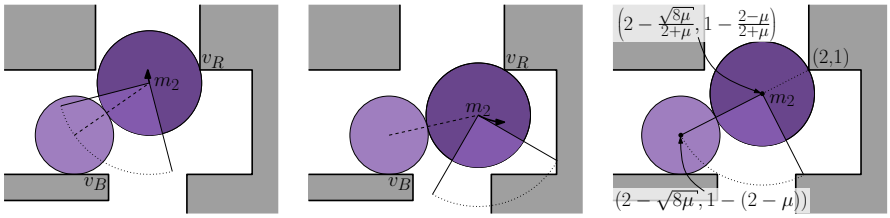
$$\begin{aligned} x(\theta) &= (1+\mu)\ln\left(\frac{\tan(\theta/2)}{\tan(\varphi/2)}\right) + (1+\mu)\Big(\cos(\theta) - \cos(\varphi)\Big) + \sin(\varphi) \\ y(\theta) &= (1+\mu)\sin(\theta) - 1 + \mu \end{aligned} \tag{2.2}$$

This motion will not always get the object to e either, as stated in the following lemma.

Lemma 2.4. *Following the hockey-stick curve from m_1 on, the object will eventually hit vertex v_R, at some point m_2. When m_2 is on or below the line $y = 1 - \frac{2-\mu}{2+\mu}$, the only possible continuations of the object's path lead it into the pocket hole on the right, and e cannot be reached.*

Proof. The curvature of the hockey-stick curve is smaller than that of the circular arc from b to e around v_L. Thus the object will reach the horizontal line through e at a point to the right of e itself. But by then point v_R would have already passed into the object's interior. Hence the object inevitably bumps into v_R at some point m_2 along the hockey-stick curve.

The y-coordinate of m_2 uniquely determines the angle θ at which the pusher can continue pushing from its contact with the bottom obstacle, as well as the push range which would push the object upwards compliantly on a circular arc



(a) (b) (c)

Figure 2.14: Being pushed along the hockey-stick curve, the object will eventually hit the outer corner vertex v_R. (a) If the object hits v_R high enough, the pusher has no trouble completing the path for the object. (b) But if the object hits v_R too low, the pusher cannot do anything but push the object into the pocket hole. (c) The cut-off point is when the line from the pusher's center through m_2 passes through v_R.

around v_R. If m_2 is high enough, θ may lie in this push range, as seen in Figure 2.14(a).

If m_2 is too low, θ will be outside of the needed push range. Having pushed the object against v_R, the pusher will still be to the left of vertex v_B, thus it cannot get below the object. Therefore, the only remaining options for pushing lead the object into the pocket hole, as seen in Figure 2.14(b).

The cut-off point where θ lies just outside the needed push range is when P's center, O's center, and vertex v_R are collinear. O's center then has coordinates $\left(2-\frac{\sqrt{8\mu}}{2+\mu}, 1-\frac{2-\mu}{2+\mu}\right)$, as illustrated by Figure 2.14(c). □

We are now ready to formulate and prove the main theorem of this section.

Theorem 2.7. *There is a μ_c, $1/3 < \mu_c < 1$, such that for all $\mu > \mu_c$, the example of Figure 2.12 admits no contact-preserving push plans for any of the possible object paths τ, while it does admit an unrestricted push plan (for some of these paths).*

Proof. From Equation (2.2) it follows that the hockey-stick curve intersects the line $y = 1 - \frac{2-\mu}{2+\mu}$ exactly when

$$\sin(\theta) = \frac{2-\mu}{2+\mu} \tag{2.3}$$

The point on this line where the object touches v_R has x-coordinate $2 - \frac{\sqrt{8\mu}}{2+\mu}$. Thus v_R will be hit when the object is on or below the line $y = 1 - \frac{2-\mu}{2+\mu}$ if and only if

$$x(\theta) \geq 2 - \frac{\sqrt{8\mu}}{2+\mu} \tag{2.4}$$

Combining Equations (2.1) through (2.4) this condition becomes

$$\ln(f(\mu)) \geq g(\mu), \tag{2.5}$$

where

$$f(\mu) = \frac{\left((2-\mu)\,(1+\mu)\;-\;\sqrt{6\mu-2}\right)\,\left(2+\mu-\sqrt{8\mu}\right)}{(2-\mu)\,\left(\mu\,(3+\mu)\;-\;(1+\mu)\,\sqrt{6\mu-2}\right)}$$

and

$$g(\mu) = \frac{\sqrt{6\mu-2}+2-\sqrt{8\mu}}{1+\mu}.$$

On the domain $\mu \in (1/3, 1)$, both g and the logarithm of f are continuous functions, and they intersect each other at exactly one $\mu = \mu_c$ (≈ 0.57173). For $\mu < \mu_c$ the function g has a greater value than the logarithm of f, and vice versa for $\mu > \mu_c$, as shown in Figure 2.15. Thus there is a μ_c such that, for all $\mu > \mu_c$, the highest possible path for the object hits v_R at a point too low to escape the pocket hole. Since any other path is below this highest possible path, no contact-preserving push plans can exist.

In contrast, for some object paths an unrestricted push plan exists for all μ between 0 and 1, including those above μ_c. Figure 2.16 illustrates one such object path and the resulting unrestricted push plan. □

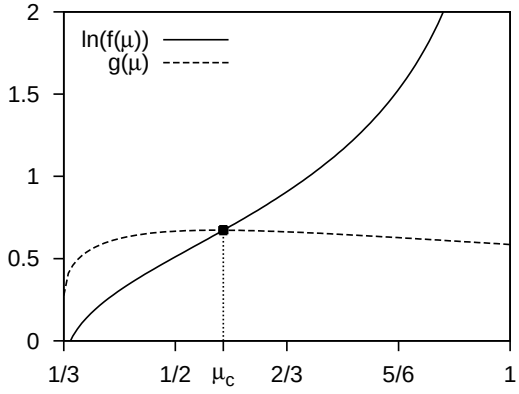


Figure 2.15: The smallest μ value for which the example of Figure 2.12 admits no contact-preserving push plan.

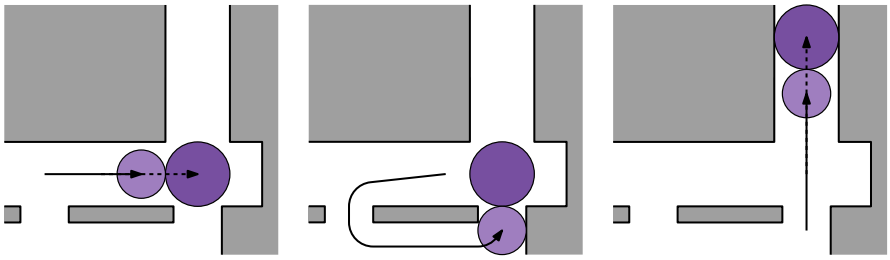


Figure 2.16: When the pusher is allowed to release the object, a push plan *does* exist.

2.5 Discussion and future research

We have studied the manipulation planning problem of a disk-shaped pusher moving a disk-shaped object along a given path, among non-intersecting line segments in the plane. We looked at the case where the pusher and object must maintain contact, as well as the case where there is no such restriction. For the contact-preserving case, we improved the running time of the only known algorithm, and gave the first algorithm to compute a shortest push plan. For the unrestricted case, we gave the first algorithm to compute a push plan at all. The running times of these algorithms are summarized in Table 2.1. Apart from being more efficient, our algorithms also handle a more general class of input paths than prior work, and can be modified to handle a more general class of obstacles as well (specifically, convex pseudodisks).

	High obstacle density		Low obstacle density	
	NH et al.	Our method	NH et al.	Our method
Prepro-cessing	$n^2 \log n$	$n \log n$ $^{(*)}$	$n^2 \log n$	$n \log n$ $^{(*)}$
Any CPPP	$kn \log n$	$kn \log n$ $^{(*)}$	$(k+n)\log(k+n)$	$(k+n)\log(k+n)$
Shortest CPPP	—	$kn \log n$ $^{(*)}$	—	$kn \log n$ $^{(*)}$
Any UPP	—	$kn \log k + kn^2 \log n$	—	$(k+n)\log(k+n) + kn$

$^{(*)}$ These entries are expected times. For the worst-case times, replace $\log n$ by $\log^2 n$.

Table 2.1: A comparison of the asymptotic running times of Nieuwenhuisen's approach and ours for computing contact-preserving push plans (CPPP) and unrestricted push plans (UPP) for a given object path.

An obvious open question is how to compute a shortest unrestricted push plan, or if the running times of our algorithms can be further improved. Additionally, one may be interested in other shapes of the pusher and/or object than mere disks. A pushing motion may then result in either or both of them rotating. The respective orientations of the pusher and object will make the configuration space higher dimensional. Lynch and Mason [59] discuss conditions under which the relative orientation of the pusher and object remain fixed. Assuming this is the case makes the problem somewhat more tractable, but our method based on a 2-dimensional configuration space would still not suffice.

Applying the configuration-space approach to the problem where only a destination for the object is given (rather than a path), is also non-trivial. While there is a straightforward analogue to our configuration-space obstacles in this 3-dimensional configuration space, there is none for our forbidden push ranges. It may be possible to use some form of constrained path finding instead, but we have not explored this possibility.

Complexity of Dynamic Point Labeling

3

Static point labeling is the task of placing textual labels next to points. This has important real-world applications. For example, in cartography one needs to place the names of cities on a map, and in a technical or medical drawing, various parts of interest need to have names attached to them. As such, static point labeling has been researched extensively, leading to experimentally validated heuristics as well as algorithms with proven approximation ratios; see Section 1.2.2 for a more extensive discussion.

With technological advances, however, modern applications of labeling have become more demanding. In maps accessed through smartphones or car navigation systems, the user's view on the map undergoes panning, rotation, and/or zooming. Similarly, technical designs and medical data are interactively explored from all angles through 3-dimensional graphics on a computer. This causes points to move, and appear or disappear from the view. In some applications the points even move, appear, and disappear on their own, rather than in response to changes in the view. Air traffic controllers, for example, monitor airplanes entering and exiting their airspace as moving points labeled with flight numbers, altitude, and velocity. In short, methods for *dynamic point labeling* are needed.

Far fewer heuristics have been published for problems of this kind, and virtually no algorithms with proven approximation ratios (see Section 1.2.3). We believe this relative lack of results for dynamic point labeling is not due to a lack of attempts. Intuitively, dynamic point labeling should be much harder than its static counterpart. In this chapter, we prove and quantify this intuition. Specifically, we show strong PSPACE-completeness of various dynamic point labeling problems, each having a static counterpart that is strongly NP-complete.

3.1 Dynamic point labeling

For a point labeling to be useful, the placed labels should be readable and it should be clear to which points they belong. This is typically achieved by regarding the labels as axis-aligned rectangles slightly larger than the text they contain, and to have two requirements on the placements: (i) each label contains its point on its boundary, and (ii) all label interiors are pairwise disjoint. As not all label positions around a point are equally desirable, often a subset of the full label boundary is selected to contain the point. This subset is typically the same for all labels, and is specified by one of the *label models* in Figure 3.1, which were introduced in Section 1.2.1. In the 1-position model one designated corner of the label must coincide with the point, in the 2-position models there is a choice of two adjacent corners, and in the 4-position model all four corners are allowed. In the 1-slider models one side of the label is designated but the label may contain the point anywhere on this side, in the 2-slider models there is a choice between two opposite sides, and in the 4-slider model the label can have the point anywhere on its boundary.

In static point labeling, one is given one of these label models along with a point set P. The *decision problem* then asks whether there exists a *static labeling* $\mathcal{L}$ that assigns labels from the label model to all points in P, without any labels overlapping each other. Since the answer is sometimes "no", we have to compromise in some way when we want to compute a useful labeling $\mathcal{L}$. In *size maximization* we compute the maximum label size at which all labels fit (possibly shrinking the labels), in *number maximization* we only label a subset of the points (as many as possible), and in *free-label maximization* we simply allow overlapping labels (trying to maximize the number of labels without overlap). For the 4-position [31, 60] and 4-slider models [50] the decision problem is strongly NP-complete (even for unit-square labels). This implies that all

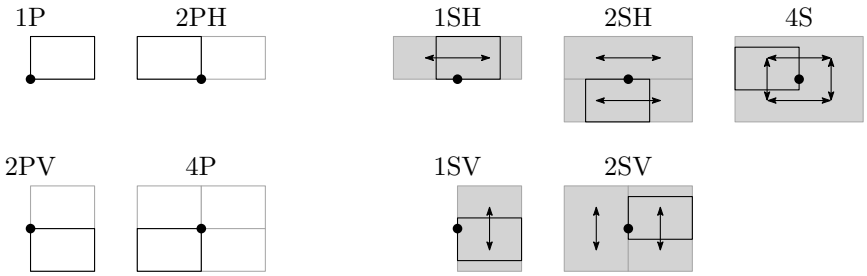


Figure 3.1: A label model specifies the allowed positions (shown in gray) for the label of a point. The fixed-position models (left) include the 1-position (1P), 2-position (2PH, 2PV), and 4-position (4P) models. The slider models (right) include the 1-slider (1SH, 1SV), 2-slider (2SH, 2SV), and 4-slider (4S) models.

of these maximization problems are strongly NP-hard for the 4-position and 4-slider models.

In dynamic point labeling we are given a label model and a *dynamic point set* P, which specifies for each point $p \in P$ an arrival time $\mathit{birth}(p)$, a departure time $\mathit{death}(p)$, and a continuous trajectory. We refer to the time interval where p is present as its *lifespan*, denoted by $\mathit{life}(p) = [\mathit{birth}(p), \mathit{death}(p)]$. Let $P(t) = \{p(t) \mid p \in P, t \in \mathit{life}(p)\}$, where $p(t)$ denotes the position of point p at time t. We then seek a *dynamic labeling* $\mathcal{L}$, that, for all t assigns a static labeling $\mathcal{L}(t)$ to the point set $P(t)$. For the 1-slider and 4-slider models we require that $\mathcal{L}$ is continuous in the sense that, if a point p has a label over a non-empty time interval, then the label must move continuously over that interval. For the fixed-position and 2-slider models we must allow labels to make "jumps". We only allow p's label to jump from position A to position B, however, if there is no candidate position C in between A and B in clockwise (or counter-clockwise) order around p. Thus, we allow horizontal and vertical jumps for the 4-position model, but no diagonal jumps. For a 2-slider model we only allow jumps from an endpoint of one slider to the "same" endpoint of the other slider. Thus, for a horizontal 2-slider we only allow vertical jumps, and only at the leftmost and rightmost positions of the labels.

The static labeling problems mentioned above can be generalized to dynamic labeling in various ways. For the decision problem, a natural generalization is the following. Does there exists a dynamic labeling $\mathcal{L}$ for a given dynamic point set P, in which all points are labeled without overlap during the entirety of a given time interval $[a, b]$? Over the next sections we shall show that this decision problem is strongly PSPACE-complete for the 4-position, 2-slider, and 4-slider models, even when the dynamic nature of P is drastically restricted and all labels are unit squares. This immediately implies that any dynamic generalization of size maximization, number maximization, and free-label maximization is strongly PSPACE-hard for these models. Moreover, we are able to show that size maximization does not admit a PTAS, unless P = PSPACE.

3.2 Structure of the reduction

To prove PSPACE-hardness of dynamic point labeling, we will reduce from *nondeterministic constraint logic (NCL)* [38], which is an abstract, single-player game. The game board is a *constraint graph*: an undirected graph with weights on both the vertices and the edges. A *configuration* of the constraint graph specifies an orientation for each of its edges. A configuration is *legal* if and only if each vertex's *inflow* (the summed weight of its incoming edges) is at least its own weight. The outflow of vertices, on the other hand, is not constrained. To make a *move* in this game is to reverse a single edge in a legal configuration such that the resulting configuration is again legal. Hearn and Demaine [38] showed that each of the following questions is PSPACE-complete for this game.

- *Configuration-to-configuration NCL*: Given two legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$, is there a sequence of moves transforming $\mathcal{C}_A$ into $\mathcal{C}_B$?

- *Configuration-to-edge NCL*: Given a legal configuration $\mathcal{C}_A$ and an orientation for a single edge e_B, is there a sequence of moves transforming $\mathcal{C}_A$ into a legal configuration $\mathcal{C}_B$ in which e_B has the specified orientation?

- *Edge-to-edge NCL*: Given orientations for edges e_A and e_B, do there exist legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$, and a sequence of moves transforming the one into the other, such that e_A has the specified orientation in $\mathcal{C}_A$ and e_B has the specified orientation in $\mathcal{C}_B$?

These decision problems remain PSPACE-complete even for planar, 3-regular constraint graphs consisting only of AND vertices and protected OR vertices. These vertex types are depicted in Figure 3.2, and are defined as follows. An *AND vertex* has a weight of 2, and its three incident edges have weights 1, 1, 2—see Figure 3.2(a). To orient the weight-2 edge away from the vertex requires both weight-1 edges to be oriented towards the vertex. An *OR vertex* and its three incident edges all have weight 2—see Figure 3.2(b). Thus at least one edge needs to be oriented towards the vertex at all times. An OR vertex is called *protected* if it has two edges that, because of constraints imposed on them by the rest of the constraint graph, cannot both be directed inward. One way to achieve this is with a triangle of two AND vertices and one OR vertex, as in Figure 3.2(c). In fact, we may assume that all OR vertices occur in such triangles. The AND vertices then share a weight-1 edge, meaning at least one of them must have an incoming weight-2 edge to satisfy its inflow constraint. This edge then cannot point into the OR vertex, making it protected.

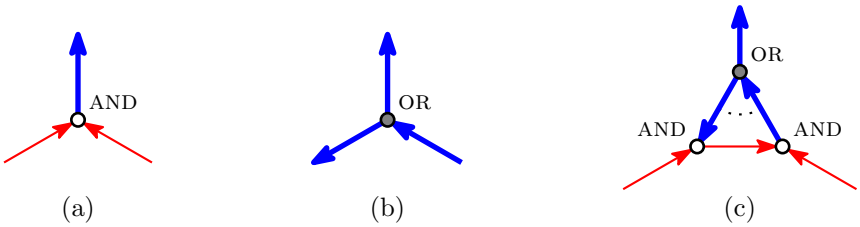


Figure 3.2: The vertex types of non-deterministic constraint logic. In this figure, and all that follow, weight-1 edges are drawn as thin, red lines and weight-2 edges are drawn as thick, blue lines. (a) An AND vertex, indicated as a white disk. (b) An OR vertex, indicated as a gray disk. (c) A way to create a protected OR vertex, where the two edges indicated by the dotted arc cannot be directed inward simultaneously.

As a first step towards proving hardness of dynamic labeling, we will show how to simulate a constraint graph G with a point set P. This will be done in such a way that labelings of P will correspond to configurations of G, and label movements will correspond to changing the orientations of edges in G. In the next section, we will then show how to use this construction to prove the PSPACE-hardness of various dynamic labeling problems.

3.2.1 High-level overview

At a high level, our construction works as follows. Given a planar constraint graph G with n vertices, we first embed it on a regular grid that has been turned by 45° relative to the coordinate axes. In this embedding G's vertices lie on grid vertices, and its edges form interior-disjoint paths along grid lines, as depicted in the center of Figure 3.3. From the structure of Hearn and Demaine's hardness proof of NCL [38] we may in fact assume that such an embedding of G is given, but otherwise we can compute one in $O(n)$ time, for example with an algorithm of Biedl and Kant [15]. The next step is to replace each vertex and edge by appropriate *gadgets* built out of points that are to receive labels. In the figure, our gadgets are depicted in the circular insets. We call the points that have been depicted with dark gray labels *blockers*, as we can consider their labels to be fixed obstacles: labeling them differently than shown will only restrict the

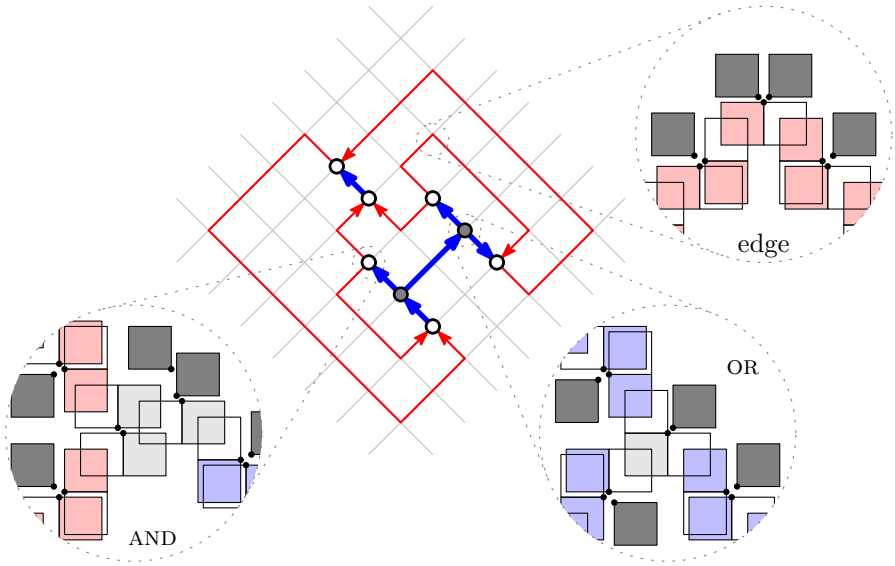


Figure 3.3: Our construction for simulating non-deterministic constraint logic with dynamic point labeling.

placement of other labels further. As is, the depicted blockers restrict the red and blue labels to two possible positions each in the 4-position model. These correspond to the two different orientations of edges. Labels "directed" *into* a vertex gadget (such as the red labels in the depicted AND gadget) correspond to edges pointing *out* of the vertex. Conversely, labels directed *out* of a vertex gadget (such as the blue labels in the depicted AND gadget) correspond to edges pointing *into* the vertex. The inflow constraints are then enforced by the light gray labels, which restrict the amount of usable space for the red and blue labels inside the vertex gadget.

3.2.2 Gadgets

Figures 3.4–3.6 show our vertex and edge gadgets in more detail. As mentioned before, the blockers restrict the colored labels marked A, B, and C to two possible positions each in the 4-position model. We call the label position closest to the center of the vertex gadget *inward*, and the other *outward*. In the figures, labels A and B are placed inward, and label C is placed outward. In the slider models, labels can take on any position in between these two extremes, but we may assume that only a very small range of positions is actually used. Consider, for example, label A in Figure 3.4. Without moving label A', we can only move A left by at most ε, or down by at most 2ε. We refer to positions for A from this range as *inward*, and define the term similarly for B and C. If (and only if) A' is moved down by at least $1-\varepsilon$, then A can move further leftward. We may

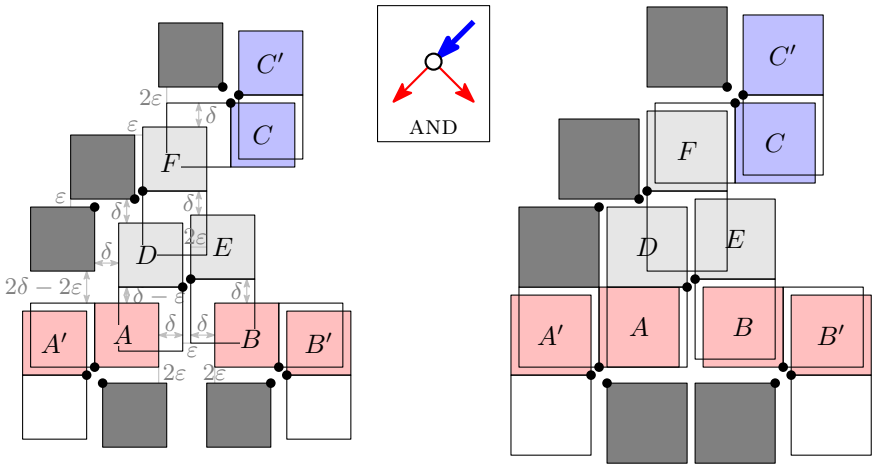


Figure 3.4: Our gadget simulating the AND vertex shown in the inset at the top. The construction works for square labels of side length 1 (left) up to $1+\delta-\varepsilon$ (right). In the figure, $\delta = 3/8$ and $\varepsilon = 1/8$.

then move A all the way to its leftmost position, and there is no reason not to do so. Thus we may define *outward* as in the 4-position model.

With this terminology in place, we shall prove that our construction faithfully simulates a constraint graph, with labels placed *inward* corresponding to edges directed *out* of a vertex, and labels placed *outward* corresponding to edges directed *into* a vertex. For this we will have to show that any overlap-free labeling of the depicted points corresponds to a legal configuration of the constraint graph, and that all legal configurations can be represented in this way. Furthermore, we need to show that a move can be made in the constraint graph if and only if the corresponding label movement can be done continuously without overlap. We shall first show this is the case locally for the vertex gadgets, and then show how our edge gadgets make this true globally.

Lemma 3.1. *The construction in Figure 3.4 satisfies the same constraints as an NCL* AND *vertex with incident edges A, B, and C, where A and B are the edges of weight* 1*. This holds for the 4-position model when $0 < \varepsilon \leqslant \delta < 1 - 3\varepsilon$, and for the 2-slider and 4-slider models when $0 < \varepsilon \leqslant \delta < 1/3 - 4\varepsilon/3$.*

Proof. As argued above we may assume the blockers are always labeled as depicted. In a labeling without overlap the labels marked A, B, and C can then only be placed inward or outward, corresponding to the opposite orientation of the respective incident edges of the AND vertex. To show that the vertex's inflow constraint is always satisfied, we must show that in an overlap-free labeling either C must be placed outward, or both A and B must be placed outward. For the 4-position model this is fairly easy to see. The depicted situation has A and B placed inward and C placed outward. Placing C inward can be done if and only if we place D, E, and F downward, which in turn necessitates placing A and B outward. For the 2-slider and 4-slider models we will show the same by analyzing the gap between F and the blocker nearest to C. To maximize this gap, first move A and B down by 2ε so they touch their blockers. With unit-square labels, D and E can then move down by $\delta + \varepsilon$ and $\delta + 2\varepsilon$, respectively. Finally, F may move down by $2\delta + 2\varepsilon$. The resulting gap above F then has size $3\delta + 4\varepsilon$, and cannot be widened any further. Since $3\delta + 4\varepsilon < 3(1/3 - 4\varepsilon/3) + 4\varepsilon = 1$, label C does not fit in this gap, meaning it cannot be placed inward. Increasing the label size will make the gap even narrower.

It remains to argue that labels can be moved if and only if the corresponding edge reversals are possible in the constraint graph. It follows from the preceding paragraph that we cannot perform any label moves not corresponding to legal moves on the constraint graph. Thus we need to show only that all legal moves can be performed as label motions. Such label motions are easily produced. We simply move D downward when A moves outward, and move D upward when A moves inward. Similarly, E should move along with B, and F with C. □

Lemma 3.2. *The construction in Figure 3.5 satisfies the same constraints as an NCL protected* OR *vertex with incident edges A, B, and C, where A and B form the protected edge pair. This holds for the 4-position model when $0 < \varepsilon \leqslant \delta < 1-3\varepsilon$, and for the 2-slider and 4-slider models when $0 < \varepsilon \leqslant \delta < 1/3-4\varepsilon/3$.*

Proof. Recall that in a protected OR vertex, the protected edges can be assumed to never both be directed into the vertex, because of the inflow constraints of neighboring AND vertices. We may therefore assume in our protected OR gadget that the labels A and B are never both placed outward. We now need to show that it is not possible to place A, B, and C all inward simultaneously. Suppose we would place all three inward. In the 4-position model, clearly at least one of them must overlap with D. To see this for the 2-slider and 4-slider models, recall that "inward" means that A, B, and C have all been displaced by at least $1-\varepsilon$ from their most outward position. With unit-square labels this leaves a gap of $2\delta+\varepsilon$ between A and C, which can be increased to $2\delta+3\varepsilon$ by moving C towards its blocker, but cannot be widened any further. Since $2\delta+3\varepsilon < 2(1/3-4\varepsilon/3)+3\varepsilon = 2/3+\varepsilon/3 < 1$, label D does not fit in this gap. Similar reasoning applies to the gap between B and C. Increasing the label size will make the gaps even narrower.

We next need to show that labels can be moved if and only if the corresponding edges can be reversed. The preceding paragraph shows that we cannot simulate illegal moves with label motions. It remains to show that all legal moves can be simulated with label motions. Moving one of the labels A, B, or C outward is always possible. To move one of them inward is only problematic if label D is currently in the way. We then need to move D out of the way first. When moving C inward this is easy, as D can always go either to the A

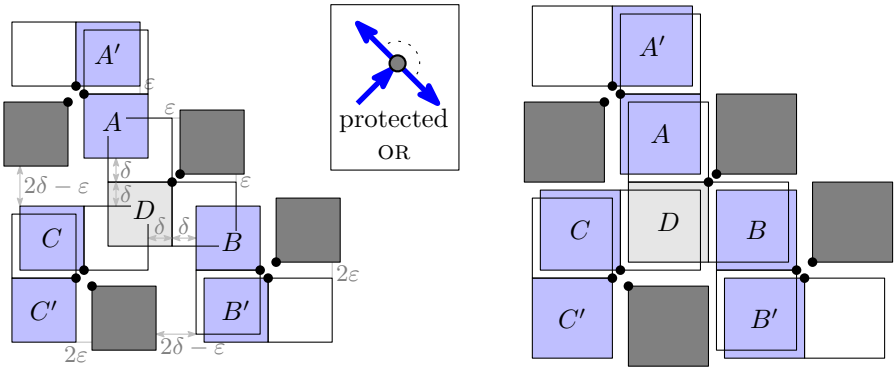


Figure 3.5: Our gadget simulating the protected OR vertex shown in the inset at the top. The construction works for square labels of side length 1 (left) up to $1+\delta-\varepsilon$ (right). In the figure, $\delta = 3/8$ and $\varepsilon = 1/8$.

side or to the B side. When moving A or B inward a problem would occur if C was already placed inward. Then D would have to "jump" over C. Note, however, that this requires both A and B to be placed outward prior to moving one of them inward. Such a scenario cannot occur because the OR vertex is protected. □

Having demonstrated working vertex gadgets, it remains to show that we can connect them over longer distances in such a way that a label moving out of one vertex gadget causes a label of another vertex gadget to move inward, and vice versa. This is done by building edge gadgets out of the pieces on the left in Figure 3.6. By interleaving bend pieces with chains of wire pieces, and varying the spacing between wire pieces appropriately, any desired polygonal chain of diagonal segments can be simulated. An example is shown on the right in the figure. By the same process an edge gadget can be connected to its two vertex gadgets, as the three colored ends of each vertex gadget are simply wire pieces. As an example, we might identify the points labeled W_1 and W_1' on the right in Figure 3.6 with the points labeled by A and A' in Figure 3.4.

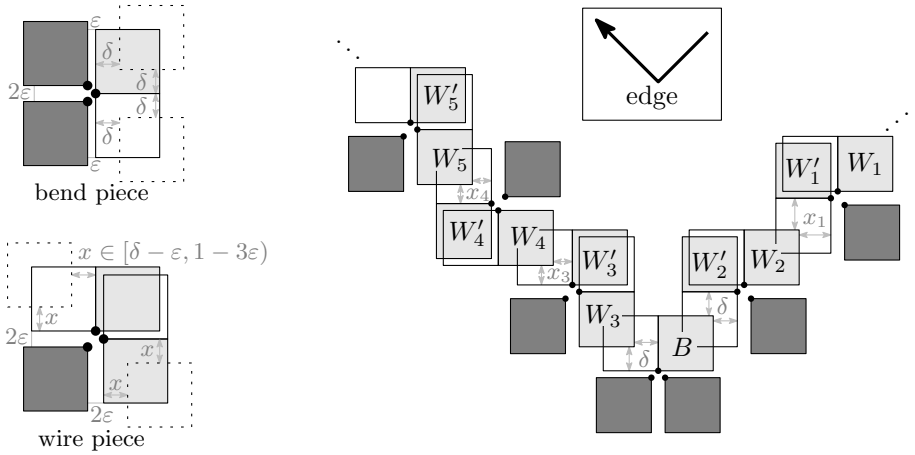


Figure 3.6: (Left) The building pieces for our edge gadgets. They may be rotated arbitrarily in 90° increments, and "connected" together along the dotted lines. (Right) An example gadget built with these pieces, simulating the edge shown in the inset at the top. The construction is the same for (red) weight-1 edges and (blue) weight-2 edges, and works for square labels of side length 1 up to $1 + \delta - \varepsilon$. In the figure, $\delta = 3/8$ and $\varepsilon = 1/8$. The distance marked x may be varied along an edge gadget in order to make it line up correctly with vertex gadgets. For the 4-position model, $x \in [\delta - \varepsilon, 1)$; for the 2-slider and 4-slider models, $x \in [\delta - \varepsilon, 1 - 3\varepsilon)$.

Lemma 3.3. *A label on one end of an edge gadget (such as the one in Figure 3.6) may only move out of its vertex gadget if the label on the other end is placed into its vertex gadget. This holds for the 4-position model when* $0 < \varepsilon \leqslant \delta < 1-3\varepsilon$*, and for the 2-slider and 4-slider models when* $0 < \varepsilon \leqslant \delta < 1/3-4\varepsilon/3$.

Proof. In the figure, W_1 is placed inward relative to a non-pictured vertex gadget above and to its right. Suppose we want to move it outward. This means W_1 moves left by more than ε, which can only happen if W_1' moves downward by at least $1-\varepsilon$. Moving W_2 down by 2ε so that it touches its blocker is not sufficient to accommodate this, because $x_1+2\varepsilon < (1-3\varepsilon)+2\varepsilon = 1-\varepsilon$. The only option is therefore to move W_2 left by at least $1-x_1 > \varepsilon$, for which W_2' has to move down by at least $1-\varepsilon$. In turn, B then has to move left by at least $1-\delta > \delta+2\varepsilon$, for which W_3 has to move up by $1-\delta > \varepsilon$, for which W_3' has to move left by $1-\varepsilon$, and so forth. Symmetric reasoning can be used to show that such a "signal" can also traverse the edge gadget in the opposite direction (from W_5' to W_1). □

Lemma 3.3 implies that an edge gadget cannot have both ends placed outward of their respective vertex gadgets. On the other hand, both ends *can* be placed inward. This corresponds to an edge of the constraint graph being directed into *neither* of its two incident vertices. To obtain a correspondence between configurations of a constraint graph and labelings of our point set we therefore modify the definition of a legal configuration to allow such "unoriented" edges. A move then either reverses an oriented edge, unorients an oriented edge, or orients an unoriented edge. This does not meaningfully change NCL: if all inflow constraints are satisfied with some unoriented edges, they remain satisfied when such edges are oriented arbitrarily.

3.2.3 Putting it all together

The results of this section now lead to the following theorem.

Theorem 3.1. *Let G be a planar constraint graph with n vertices, and let ε and δ be two real numbers with $0 < \varepsilon \leqslant \delta < 1-3\varepsilon$. One can then construct a point set $P = P(G, \delta, \varepsilon)$ that has the following properties for any $s \in [1, 1+\delta-\varepsilon]$:*

- *the size of P, and the coordinates of the points in P, are polynomially bounded in n,*
- *for any legal configuration of G there is an overlap-free static labeling of P with $s \times s$ square labels in the 4-position model and vice versa,*
- *there is a sequence of moves transforming one legal configuration of G into another if and only if there is a dynamic labeling transforming the corresponding static labelings of P into each other.*

When $\delta < 1/3 - 4\varepsilon/3$, the same results hold for the 2- and 4-slider models.

Proof. As mentioned above, the given constraint graph G may be assumed to be embedded on a grid, with its vertices on grid vertices and its edges being interior-disjoint paths along grid lines. Such a structure follows from Hearn and Demaine's hardness proof of NCL [38], but we could otherwise achieve it in $O(n)$ time, for example with an algorithm of Biedl and Kant [15]. We turn this grid by 45° relative to the coordinate axes, and replace G's vertices and edges by the appropriate gadgets of Lemma 3.2 through 3.3, as was illustrated in Figure 3.3. If we choose the scale of the grid and the spacing between the wire pieces of edge gadgets appropriately, then the gadgets will fit together perfectly. The union of the points making up the gadgets forms the desired set P.

That the second and third claims hold for this construction follows from Lemma 3.2 through 3.3. To see that the first claim holds, note that the coordinates of all points within our gadgets are linear combinations of 1, δ, and ε, with integer coefficients. Furthermore, the amount of freedom in the alignment of wire pieces allows us to maintain this property when connecting gadgets together. Thus we need only show that a polynomial number of wire pieces is used, so that none of these integer coefficients can become super-polynomial. This follows immediately when using Biedl and Kant's algorithm, as they lay out G on an $n \times n$ grid with at most $2n + 2$ edge bends in total. □

3.3 Hardness of dynamic point labeling

In this section we will define a number of dynamic point-labeling problems. The input to all of them is a dynamic point set P, which specifies for each point $p \in P$ an arrival time $\mathit{birth}(p)$ and a departure time $\mathit{death}(p)$, as well as a continuous trajectory. In some problems these changes to the point set may be fairly arbitrary, in others they are the result of the user panning, rotating, or zooming their viewport of a static point set. In all cases we seek a dynamic labeling $\mathcal{L}$ of P for a given time interval $[a, b]$. That is, $\mathcal{L}(t)$ must be a static labeling of $P(t)$ for all $t \in [a, b]$, and each of $\mathcal{L}$'s labels must move continuously over time.

Various optimization questions may be formulated for this setting. We may disallow label overlap entirely, and then seek a dynamic labeling that labels as many points as possible for as long as possible. Alternatively, perhaps we wish to label all points at all times, and wish to have as little label overlap as possible. Whatever the case may be, labeling all points at all times without any overlap is likely the most desirable outcome. Unfortunately, we will see that deciding whether such a solution exists is already strongly PSPACE-complete, even if we drastically restrict the dynamic nature of the point set. Thus, all optimization problems of this kind are strongly PSPACE-hard.

3.3.1 Transforming one labeling into another

As a warm-up, we start with a very simple PSPACE-hardness proof. The first problem we consider does not concern a dynamic point set or viewport at all. Instead, we are given two distinct static labelings $\mathcal{L}(a)$ and $\mathcal{L}(b)$ for one static point set P. We then want to find a dynamic labeling $\mathcal{L}$ that transforms $\mathcal{L}(a)$ into $\mathcal{L}(b)$ over the time interval $[a, b]$ by continuous label movement, without any labels ever overlapping.

Theorem 3.2. *The following decision problem is strongly PSPACE-hard for the 4-position, 2-slider, and 4-slider label models.*

> ***Given:*** *A static point set P, numbers a and b with $a < b$, and two static labelings $\mathcal{L}(a)$ and $\mathcal{L}(b)$ of P using non-overlapping unit-square labels.*
> ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P with $\mathcal{L}(a)$ and $\mathcal{L}(b)$ as given, which labels all points with non-overlapping unit-square labels over the time interval $[a, b]$.*

Additionally, unless $P = PSPACE$, the maximum label size for which there is such an $\mathcal{L}$ cannot be $(4/3 - \varepsilon')$-approximated in polynomial time for any $\varepsilon' > 0$. For the 4-position model, it cannot even be $(2 - \varepsilon')$-approximated.

Proof. To show PSPACE-hardness, we reduce from configuration-to-configuration NCL. Thus, we are given a constraint graph G with two legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$ of its edges, and want to decide whether there is a sequence of moves transforming $\mathcal{C}_A$ into $\mathcal{C}_B$. By Theorem 3.1, we may construct a point set $P = P(G, \delta, \varepsilon)$ with two valid static labelings $\mathcal{L}(a)$ and $\mathcal{L}(b)$ corresponding to configurations $\mathcal{C}_A$ and $\mathcal{C}_B$. The desired dynamic labeling then exists if and only if there is a sequence of moves transforming $\mathcal{C}_A$ into $\mathcal{C}_B$, and deciding the latter is PSPACE-hard. All coordinates are polynomially bounded in the size of G, making the decision problem strongly PSPACE-hard. The construction works for identical square labels with a side length in the range $[1, 1 + \delta - \varepsilon]$. We must have $0 < \varepsilon \leqslant \delta < 1 - 3\varepsilon$ for the 4-position model, so pick $\varepsilon = \varepsilon'/4$ and $\delta = 1 - \varepsilon'$ to obtain the claimed hardness-of-approximation result. For the 2-slider and 4-slider models $\delta < 1/3 - 4\varepsilon/3$ must hold, so pick $\delta = 1/3 - \varepsilon'$ instead. □

3.3.2 Labeling dynamic point sets

Now suppose only one of $\mathcal{L}(a)$ and $\mathcal{L}(b)$ is given, and we are free to choose the other; does there exist a dynamic labeling then? For a static point set the answer is trivially "yes": simply set $\mathcal{L}(a) = \mathcal{L}(b) = \mathcal{L}(t)$ for all $t \in [a, b]$. But when labeling a dynamic point set where points move, or in which points are added and removed, this question becomes hard.

Theorem 3.3. *The following decision problem is strongly PSPACE-hard for the 4-position, 2-slider, and 4-slider label models.*

Given: *A dynamic point set P (with given trajectories, arrival times, and departure times), numbers a and b with $a < b$, and a static labeling $\mathcal{L}(a)$ for $P(a)$.*
Decide: *Whether there exists a dynamic labeling $\mathcal{L}$ for P respecting $\mathcal{L}(a)$ that labels all points with non-overlapping unit-square labels over the time interval $[a, b]$.*

Additionally, unless P = PSPACE, the maximum label size for which there is such an $\mathcal{L}$ cannot be $(4/3-\varepsilon')$-approximated in polynomial time for any $\varepsilon' > 0$. For the 4-position model, it cannot even be $(2-\varepsilon')$-approximated.

All of the above remains true when

- *all points are stationary, and during $[a, b]$ one point is removed and one point is added,*
- *no points are added or removed, and all points move at the same, constant speed along parallel (but possibly opposite), straight-line trajectories, or*
- *no points are added or removed, and all points but one are stationary.*

Proof. To show PSPACE-hardness when $\mathcal{L}(a)$ is given, we reduce from configuration-to-edge NCL. Thus we are given a constraint graph G with a legal starting configuration $\mathcal{C}_A$ and the goal orientation of a single edge e_B, and want to decide whether there is a sequence of moves on G starting from $\mathcal{C}_A$ that will result in e_B having its specified orientation. By Theorem 3.1, we may construct a point set $P = P(G, \delta, \varepsilon)$ and a valid static labeling $\mathcal{L}(a)$ corresponding to configuration $\mathcal{C}_A$. Now pick one blocker $q \in P$ in the edge gadget for e_B,

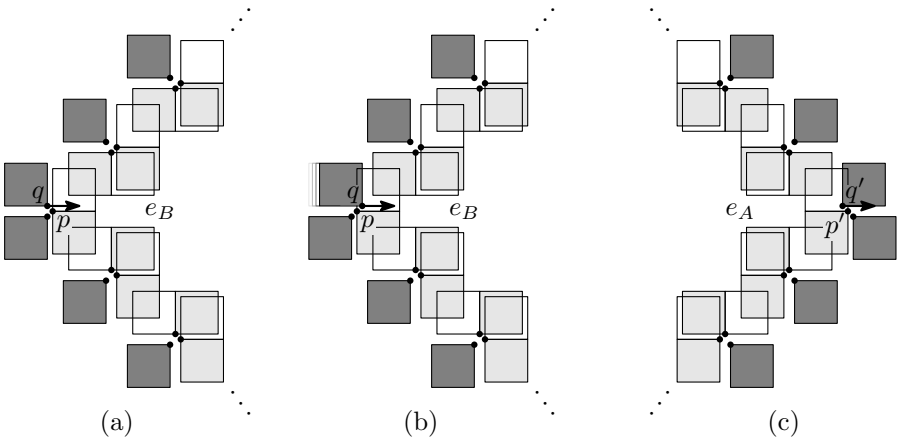


Figure 3.7: Our reduction from edge-to-edge NCL to dynamic labeling of moving points. (a) The modified gadget for edge e_B at time a. (b) The modified gadget for edge e_B at time b. (c) The modified gadget for edge e_A at time a.

and suppose $p \in P$ is the point for which q blocks some label candidates. We now make q move at a constant speed of $v = 2\varepsilon/(b-a)$ towards the nearest non-blocked candidate of p, as in Figure 3.7(a). Figure 3.7(b) shows the end result at time b: the edge gadget has become constrained to a single orientation around p. Thus there is a sequence of moves on $\mathcal{C}_A$ resulting in e_B having the specified orientation if and only if there is a dynamic labeling $\mathcal{L}$ for this dynamic point set starting at $\mathcal{L}(a)$. The same result may be obtained by moving q at speed $v/2$ and moving all other points at speed $v/2$ in the opposite direction. Alternatively, we may remove the point q from P and re-insert it at its modified location. In conclusion, all the stated problem formulations are PSPACE-hard.

To prove the remaining claims (that the decision problem is strongly hard, and the hardness-of-approximation results) proceed in the same way as in Theorem 3.2. □

By symmetry, the hardness claims in Theorem 3.3 remain true when $\mathcal{L}(b)$ is given instead of $\mathcal{L}(a)$, as one can simply exchange the roles of a and b in the proof. When neither $\mathcal{L}(a)$ nor $\mathcal{L}(b)$ is given, so that both can be chosen freely, the problem does not become much easier, either.

Theorem 3.4. *The following decision problem is strongly PSPACE-hard for the 4-position, 2-slider, and 4-slider label models.*

> ***Given:*** *A dynamic point set P (with given trajectories, arrival times, and departure times), and numbers a and b with $a < b$.*
> ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels all points with non-overlapping unit-square labels over the time interval $[a, b]$.*

Additionally, unless P = PSPACE, the maximum label size for which there is such an $\mathcal{L}$ cannot be $(4/3-\varepsilon')$-approximated in polynomial time for any $\varepsilon' > 0$. For the 4-position model, it cannot even be $(2-\varepsilon')$-approximated.

All of the above remains true when

- *all points are stationary, and during $[a, b]$ two points are removed and two points are added,*
- *no points are added or removed, and all points move at the same, constant speed along parallel (but possibly opposite), straight-line trajectories, or*
- *no points are added or removed, and all but two points are stationary.*

Proof. To show PSPACE-hardness, we reduce from edge-to-edge NCL. Thus we are given a constraint graph G with orientations for two edges e_A and e_B, and want to decide whether there is a sequence of moves on G starting from a legal configuration $\mathcal{C}_A$ where e_A has its specified orientation and ending in a legal configuration $\mathcal{C}_B$ where e_B has its specified orientation. We proceed in the same way as in Theorem 3.3, but now modify blockers in *two* edge gadgets. For e_B we make the blocker q move into the edge gadget at speed v as before, for e_A we start the blocker q' inside the edge gadget and make it move outward

at speed v, as in Figure 3.7(c). We pick a slightly faster speed $v = 3\varepsilon/(b-a)$, so that e_A is constrained only during the time interval $[a, a+\Delta)$, and e_B only during the time interval $(b-\Delta, b]$, where $\Delta = (b-a)/3$. If we take care in how the edge gadgets are laid out on the grid, then we can always ensure that the two blockers move in the same direction (as they do in the figure). We may then reduce their speeds to $v/2$, and move all other points at speed $v/2$ in the opposite direction. Alternatively, at time $a+\Delta$ we may remove the one blocker and re-insert it at its new location, and do the same for the other blocker at time $b-\Delta$. The remaining claims are proved as in preceding theorems. □

3.3.3 Labeling under panning, zooming, and/or rotation

In interactive mapping applications, users are presented with a rectangular *viewport* V showing a portion of a larger map. By dynamically panning, rotating, and/or zooming the map, the user controls which portion of the map is displayed at any given time. The task of labeling the points inside V can be seen as a special case of labeling dynamic point sets. Continuous panning, rotation, and zooming of the map may cause points to enter or leave V at its boundary, and causes all points within V to move on continuous trajectories. We will require only the points inside V to be labeled, and these labels must be fully contained in V. Points outside of V need not be labeled, but we may wish to do so in order to ensure a smooth dynamic labeling. Otherwise a label would have to instantly appear or disappear whenever its point hits the boundary of V. Regardless of whether we allow such "popping" of labels, it turns out that the question of whether a dynamic labeling can label all points in the viewport without overlap is strongly PSPACE-hard. We prove this first for panning.

Theorem 3.5. *The following decision problem is strongly PSPACE-hard for the 4-position, 2-slider, and 4-slider label models.*

> ***Given:*** *A closed rectangle V panning along a given trajectory, a set of points P, and two numbers a and b with $a < b$.*
> ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels $P \cap V(t)$ with non-overlapping unit-square labels inside $V(t)$ for all $t \in [a, b]$.*

This is true regardless of

- *whether panning occurs on a straight line at constant speed or on some other trajectory,*
- *whether points on the boundary of V may instantly lose/gain their labels or not, and*
- *whether the labelings $\mathcal{L}(a)$ and/or $\mathcal{L}(b)$ are given or not.*

Proof. We prove PSPACE-hardness for the 4-slider model, with neither $\mathcal{L}(a)$ nor $\mathcal{L}(b)$ being given. The remaining results can then be derived by similar techniques as in previous theorems. Our reduction is from edge-to-edge NCL,

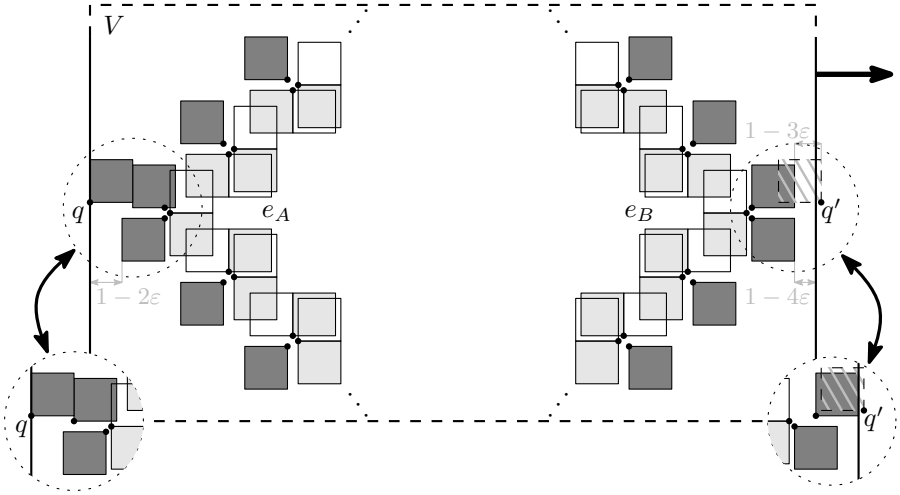


Figure 3.8: Our reduction from edge-to-edge NCL to dynamic labeling under panning. The main figure shows the construction for the 4-slider model, the circular insets in the bottom left and right show how to modify the construction for the 4-position model.

where we are given a constraint graph G and orientations for two of its edges e_A and e_B. We construct the point set $P = P(G, \delta, \varepsilon)$ as usual, but this time we lay it out on the grid in such a way that e_A is on the far left and e_B is on the far right, as in Figure 3.8. While this is always possible, it may introduce edge crossings. However, Hearn and Demaine show [38] how to construct a planar "cross-over" in NCL that functions as two crossing edges. This cross-over uses only AND and protected OR vertices, so we may emulate it using our gadgets.

Next, we add two additional points q and q', with q to the left of e_A by $1-2\varepsilon$ and q' to the right of e_B by $1-3\varepsilon$, as in Figure 3.8. We construct a viewport rectangle V with its left side containing q, and its right side ε to the left of q'. This forces us to label q in a way that constrains e_A to a single orientation. Now suppose we pan V to the right, which moves all points to the left. This lifts this constraint on e_A, as we may then move q's label out of V (either immediately, or over a short time interval). Panning may continue for a distance of $1-\varepsilon$ before disturbing the functioning of the edge gadget for e_A. However, already after moving by ε are we required to start labeling point q' in a way that constrains e_B to a single orientation. Thus there exists a dynamic labeling of P for this panning motion if and only if there exists a sequence of moves in G starting with e_A in its specified orientation and ending with e_B in its specified orientation. This makes the problem PSPACE-hard for the 4-slider model. For a horizontal 2-slider model we may use the exact same construction;

for a vertical 2-slider model we simply rotate the construction by 90°, placing e_A and e_B at the top and bottom. For the 4-position model, the construction needs a small modification so that labels always contain their point on a corner. This modification is depicted in the circular insets of Figure 3.8.

The cases that either $\mathcal{L}(a)$ or $\mathcal{L}(b)$ is given are similar, using a reduction from configuration-to-edge NCL that modifies only one edge gadget. If both $\mathcal{L}(a)$ and $\mathcal{L}(b)$ are specified, the problem is already hard without any point ever touching the viewport, as we saw in Theorem 3.2. Strong hardness may be proved as in Theorem 3.2. □

When panning, all points move in the same direction along parallel trajectories at identical speeds. When rotating, however, the points move on concentric circles centered on the point c at the center of the viewport V. The speed at which a point p moves is proportional to the distance $|p - c|$. Our hardness reduction for panning can be modified for the case of rotation as follows.

Theorem 3.6. *The following decision problem is strongly PSPACE-hard for the 4-position, 2-slider, and 4-slider label models.*

Given: *A closed rectangle V with center c, a set of points P rotating around c, and two numbers a and b with $a < b$.*
Decide: *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels $P(t) \cap V$ with non-overlapping unit-square labels inside V for all $t \in [a, b]$.*

This is true regardless of

- *whether the point set is rotated back and forth arbitrarily, or only in a single direction at constant speed,*
- *whether points on the boundary of V may instantly lose/gain their labels or not, and*
- *whether the labelings $\mathcal{L}(a)$ and/or $\mathcal{L}(b)$ are given or not.*

Proof. Consider again the reduction from NCL to dynamic labeling under panning that was depicted in Figure 3.8. Now suppose V has been constructed so that q and q' both lie below its center c. Rotating P clockwise around c then moves q out of V, and q' towards V. This also changes the horizontal and vertical distances between the points of the gadgets. As long as these changes are less than ε, however, all gadgets will still work. Thus, a sufficiently small rotation will not disturb our reduction. To ensure q' ends up inside V during this rotation, decrease the initial distance between V and q' appropriately by moving the right side of V slightly farther to the right. With these observations we may now prove all claims similarly as in Theorem 3.5. □

When zooming, each point $p \in P$ moves on the ray from c through p, at a speed proportional to the distance $|p - c|$. When zooming out the points move towards c, when zooming in the points move away from c. This adds an asymmetry to the problem not present for panning or rotation: it now matters *which*

static labeling is given. Suppose during $[a, b]$ the view is maximally zoomed out at time $t \in [a, b]$, and that it is the labeling $\mathcal{L}(t)$ for $P(t)$ that is given. Then the answer to the decision problem is trivially "yes", since zooming in further cannot "invalidate" this given labeling: all inter-point distances increase and no new points can enter V. If we are not given any static labelings to adhere to, then the problem is "merely" NP-complete. We simply determine the time $t \in [a, b]$ at which the view is maximally zoomed out, and then solve the corresponding static labeling problem on $P(t)$. But when given a single static labeling, and not the one where the view is maximally zoomed out, then the problem is PSPACE-hard. (Of course when both $\mathcal{L}(a)$ and $\mathcal{L}(b)$ are given the problem is also PSPACE-hard, as this was already proven in Theorem 3.2.)

Theorem 3.7. *The following decision problem is strongly PSPACE-hard for the 4-position, 2-slider, and 4-slider label models.*

> ***Given:*** *A closed rectangle V with center c, a set of points P scaling around c, numbers a and b with $a < b$, and a static labeling $\mathcal{L}(c)$ of $P(c)$ with $c \in \{a, b\}$. Here c must not be the time at which P's scaling factor is minimal during $[a, b]$.*
> ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels $P(t) \cap V$ with non-overlapping unit-square labels inside V for all $t \in [a, b]$.*

This is true regardless of

- *whether the point set is scaled back and forth arbitrarily, or only in a single direction at constant speed, and*
- *whether points on the boundary of V may instantly lose/gain their labels or not.*

Proof. Consider yet again the reduction from NCL to dynamic labeling under panning that was depicted in Figure 3.8. Suppose we remove the point q' from the construction, and then start zooming in. This will cause q to move out of V, and the constraint on e_A will be lifted. Conversely, suppose we remove point q from the construction, and then start zooming out. This will cause q' to move into V, adding a constraint on e_B. Thus we can reduce configuration-to-edge NCL to dynamic labeling under zooming. As with rotation, we will have to be careful that the changes to inter-point distances do not disturb the gadgets. The solution is the same as it was then: move the right side of V closer to q' so that we do not have to zoom so far as to disturb the gadgets. □

3.3.4 Membership in PSPACE

We have shown that even very restricted variants of dynamic point labeling are PSPACE-hard. Now, we shall show that a very general version of dynamic point labeling is in PSPACE. Here points may arrive and depart at will over the interval $[a, b]$, they may move on arbitrary algebraic trajectories of bounded degree,

and their labels can be rectangles of arbitrary sizes that need not be identical. Since this general variant encompasses all previously mentioned variants, this implies that all of them are PSPACE-complete.

Theorem 3.8. *The following decision problem is in PSPACE for the 4-position, 2-slider, and 4-slider label models.*

> ***Given:*** *A dynamic point set P (with given arrival times, departure times, bounded-degree algebraic trajectories, and label dimensions) and numbers a and b with $a < b$.*
> ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels all points with non-overlapping axis-aligned rectangles of the specified dimensions over the time interval $[a, b]$.*

The above is true regardless of whether the labelings $\mathcal{L}(a)$ and/or $\mathcal{L}(b)$ are given or not.

Proof. We shall describe a non-deterministic algorithm running in polynomial space, showing that the problem is in NPSPACE. Since PSPACE=NPSPACE by Savitch's theorem [86], this then yields the claimed result. We shall show this for the 4-slider model first, while assuming that $\mathcal{L}(a)$ and $\mathcal{L}(b)$ are *not* given, and that no points arrive or depart between times a and b. The minor modifications needed for the other cases and label models will be explained at the end of the proof.

A key idea for our algorithm is the notion of a *canonical labeling*, defined as follows. Draw vertical lines through the left and right sides of all labels. We say a label is *canonical along the x-axis* if it is placed entirely to the left or right of its point, or if its left or right side lies on a vertical line defined by another label which is canonical along the x-axis. We define a label to be *canonical along the y-axis* in an analogous fashion. A label is called canonical if its position is canonical along both axes. A static labeling is called canonical if all of its labels are canonical (see Figure 3.9).

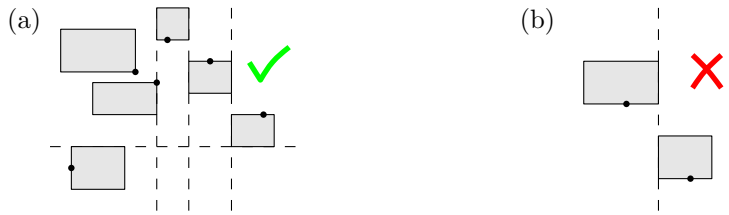


Figure 3.9: (a) An example of a canonical labeling. (b) Something that is *not* a canonical labeling: in a sequence of labels whose sides are aligned, at least one must be in an extreme position along that axis.

A canonical static labeling can be represented combinatorially by storing a pair of symbolic coordinates for each label. With n labels, there are at most $2+4(n-1) = 4n-2$ possible x-positions for each label: to the left or to the right of its point, or with its left or right side aligned with the left or right side of another label. Similarly, there are at most $4n-2$ possible y-positions. Our algorithm starts out by non-deterministically guessing such a combinatorial structure for a canonical static labeling $\mathcal{L}(t_1)$, where $t_1 = a$. Then, it computes the time $t_2 > t_1$ of the first *event*, that is, the first time that two horizontal lines or two vertical lines defined by the labels cross each other. In addition, it guesses a sequence of zero or more *label moves*, which are to be applied to $\mathcal{L}(t_1)$ in sequence between times t_1 and t_2. Each label move specifies a label and a direction (either clockwise or counter-clockwise), and applying it to a labeling means moving the label in the specified direction until it reaches a new canonical position. A label move does not have to specify a speed for the label, or a subinterval of (t_1, t_2) at which it occurs. For the combinatorial structure of the labeling it makes no difference if we assume that all label moves happen instantly one after another at unspecified times between t_1 and t_2. Having applied the label moves, the combinatorial structure of a new canonical labeling $\mathcal{L}(t_2)$ results, for which the next event time $t_3 > t_2$ is computed, and a new sequence of label moves is guessed. We continue in this fashion until reaching an event time $t_m \geqslant b$ for some m. The result is a sequence of canonical labelings at event times, interspersed with sequences of label moves. We know this sequence is finite, because: 1) the event times correspond to distinct zeroes of a polynomial that is a function of the point trajectories, and 2) any canonical labeling can be reached from any other in a finite number of label moves. Such a sequence will be referred to as a *canonical dynamic labeling*, and we claim that it exists if and only if there also exists a proper dynamic labeling.

To prove the claim in the "only if" direction, suppose that $\mathcal{L}$ is a canonical dynamic labeling for P. To make $\mathcal{L}$ a proper dynamic labeling, all that is needed is to assign disjoint and non-empty intervals of time to all its label moves. As the event times are distinct, this can always be done, for example by dividing the interval between two events evenly among its label moves.

To prove the claim in the "if" direction, suppose there is a dynamic labeling $\mathcal{L}$ for P over the interval $[a, b]$. Let $t_1 = a$ and "round" the static labeling $\mathcal{L}(t_1)$ to a canonical one by going through the points one-by-one from left to right, and for each point with a non-canonical label doing the following. First move its label continuously to the left, stopping when the label hits a vertical line defined by another label which is already canonical, or when the label is completely to the left of its point. This makes the label canonical along the x-axis. Now move the label upward in an analogous fashion until it is canonical along the y-axis. Having made $\mathcal{L}(t_1)$ canonical in this fashion, let $t_2 > t_1$ denote the first event time for this labeling. We then round $\mathcal{L}(t_2)$ to a canonical labeling in the same fashion, let $t_3 > t_2$ denote *its* first event, and continue on until reaching $t_m \geqslant b$ for some m. We now have a sequence of canonical labelings, and only

need to show that we can intersperse them with label moves to transition from one to another without label overlap. To this end, consider the continuous label movement that occurred in the original dynamic labeling $\mathcal{L}$ during the interval (t_i, t_{i+1}), for some $i \in \{1, \ldots, m-1\}$. Now prepend the label motions used in the rounding of $\mathcal{L}(t_i)$ and append the label motions used in the rounding of $\mathcal{L}(t_{i+1})$. This results in overlap-free label motions between the two canonical labelings $\mathcal{L}(t_i)$ and $\mathcal{L}(t_{i+1})$. From that we can compute a sequence of discrete label moves by determining the order in which labels cross horizontal or vertical lines in this motion.

To conclude the proof, we have to show how all of the above can be modified for the remaining cases allowed by the problem definition.

Firstly, we shall deal with the other two label models. The case of the 4-position model is actually easier than that of the 4-slider model. In the 4-position model any static labeling is canonical. Transitions between them involve moving each label one position clockwise, counter-clockwise, or not at all, and may be done instantly. A canonical dynamic labeling is therefore automatically also a proper dynamic labeling. The case of the 2-slider model is simply a combination of the other two: it behaves like the 4-slider model along one axis, and like the 4-position model along the other.

Secondly, we have assumed so far that neither $\mathcal{L}(a)$ nor $\mathcal{L}(b)$ was given, and had to be guessed non-deterministically. If either or both are given, however, the procedure is much the same. If $\mathcal{L}(a)$ is given, then we simply use *it* instead of our own guess. When guessing the label moves before the first event, the label positions as given by $\mathcal{L}(a)$ will then also count as canonical. Similarly, if $\mathcal{L}(b)$ is given, the final label moves are guessed in a way that the label positions of $\mathcal{L}(b)$ count as canonical.

Lastly, we have assumed that no points arrive or depart in the interval (a, b). Suppose points *do* arrive and/or depart, at times $a = e_1, e_2, \ldots, e_\ell = b$. Simply apply the above separately on each of the consecutive time intervals $[e_1, e_2], [e_2, e_3], \ldots, [e_{\ell-1}, e_\ell]$. The static labeling $\mathcal{L}(e_i)$ for each $i \in \{2, \ldots, \ell-1\}$ is then computed as output for the interval $[e_{i-1}, e_i]$ and used as input for the interval $[e_i, e_{i+1}]$. □

3.4 Discussion and future research

We have examined the complexity of dynamic point labeling. For a set of points where points may be added or removed over time, and where the points present move along continuous trajectories, we seek a smooth function from time to static point labelings. For the 4-position, 2-slider, and 4-slider models we have shown that deciding whether there exists such a dynamic labeling in which no labels ever overlap is strongly PSPACE-complete. In addition, finding the maximum label size at which such a labeling does exist admits no PTAS unless P = PSPACE. For the 4-position model a 2-approximation is the best that can

be hoped for, and for the other models a 4/3-approximation. The PSPACE-completeness results also apply for the special case of dynamic point labeling in which the point set is panned, rotated, or zoomed inside a fixed viewport. For this special case our constructions have a lot less "wiggle room" so that we were not able to obtain similar hardness-of-approximation results. The wiggle room is still non-zero, however, meaning that a PTAS can still not exist unless P = PSPACE.

It remains to examine the complexity of other label models, specifically the 1- and 2-position models, as well as the 1-slider model. For static labeling, the decision problem can easily be solved in polynomial time for these label models. On the other hand, the number-maximization problem is NP-hard even for static labeling in the 1-position model. For dynamic labeling perhaps the complexity landscape is similar. This would suggest that perhaps we cannot prove PSPACE-completeness of the decision problem for these label models, but that we can prove some optimization problems PSPACE-hard for them.

Most importantly, perhaps, is the continued pursuit of approximation algorithms for such optimization problems in dynamic point labeling. So far, this task has seen little success. Over the next two chapters we develop an algorithm for which we can give *some* guarantees about the quality of the produced dynamic labeling, but a true approximation ratio remains elusive.

Free-Label Maximization for Static Points

4

Ideally in a point labeling we want to assign labels to all points, with no labels overlapping each other. As this is not always possible, we have to compromise in one way or another. If we want to disallow label overlap, then we could shrink the labels, or remove some of them. The *size maximization problem* asks us to label all points with non-overlapping labels of the largest possible size. The *(weighted) number-maximization problem* asks us to label a maximum-cardinality (maximum-weight) subset of the points, with non-overlapping labels of *given* dimensions. Of course, another option is to simply allow label overlap. The *free-label-maximization problem* asks us to label *all* points with labels of given dimensions. We then want to maximize the number of *free* labels, that is, labels not overlapping any other labels.

We proved in Chapter 3 that each of these problems is PSPACE-hard when one wants to label a dynamic point set. Thus we will have to settle for approximation algorithms. In this chapter, we develop approximation algorithms for free-label maximization on static points. We do this for two reasons. Firstly, labeling all points with labels of fixed dimensions can be a natural fit for applications in which points move (see Section 1.2.3). In European regulations for air-traffic control it is even the required way of labeling the points [25]. We will use the algorithms on static points of this chapter to develop an algorithm for labeling moving points in the next chapter. Secondly, we fill a gap in the static point labeling literature. While the size-maximization and (weighted) number-maximization problems have been studied extensively for static points (see Section 1.2.2), free-label maximization had not been studied at all.

For unit-square labels we achieve a PTAS for this problem, as well as an $O(n \log n)$-time constant-factor approximation algorithm, for all of the fixed-position and slider models. The approximation ratio of the latter algorithm is 6 for the 1-slider model, 7 for the 2-position model, 22 for the 4-position and 2-slider models, and 32 for the 4-slider model. Through a suitable scaling

transform our results also apply if all labels are translates of a fixed rectangle. We will only discuss our results for the 2PH, 4P, 1SH, 2SV, and 4S label models. The algorithms and proofs for the other listed models (2PV, 1SV, 2SH) can be derived through symmetry. We assume throughout that no two points have the same x- or y-coordinate. This assumption is not essential, but it makes our exposition simpler.

4.1 Constant-factor approximations for unit squares

Let P be a set of n points in the plane. We wish to label each point in P with a unit-square label in such a way that the number of free labels is maximized. Next we describe a generic algorithm for this problem. It can be applied to any label model, but the approximation factor and implementation details depend on the model used.

The algorithm, which we call GreedySweep, processes the points from left to right, labeling them one by one. Whenever possible it produces a "freeable" label. We call a label or label candidate L *freeable* if it is free *at the time* and we know it will *remain* free as we continue labeling the remaining points. More formally, suppose GreedySweep has labeled points $p_1, \ldots, p_{i-1} \in P$ so far, and has now arrived at point $p_i \in P$. We denote the set of all label candidates for $p_i \in P$ according to the label model being used by $\textsc{Cand}(p_i)$. A label candidate $L \in \textsc{Cand}(p_i)$ is freeable if none of the previously placed labels intersect L, and every point still to be labeled has at least one label candidate that intersects neither L nor any previously placed freeable label. A label is freeable if it was the result of picking a freeable label candidate as that point's label. We denote by $\textsc{gs}(p_i)$ the label assigned to point $p_i \in P$ by the algorithm.

Algorithm GreedySweep(P, Cand)
1. Let $p_1, \ldots, p_n$ be the points in P, sorted in order of increasing x-coordinate.
2. **for** $i \leftarrow 1$ **to** n
3. **do if** p_i has a freeable label candidate
4. **then** Let $L \in \textsc{Cand}(p_i)$ be a leftmost freeable label candidate, with ties broken arbitrarily, and set $\textsc{gs}(p_i) \leftarrow L$.
5. **else** Let $L \in \textsc{Cand}(p_i)$ be a leftmost label candidate (with ties broken arbitrarily) that does not intersect any freeable labels that have already been placed. Note that such an L always exists, by definition of freeable. Set $\textsc{gs}(p_i) \leftarrow L$.

Later, in Section 4.1.3, we will show how to implement the algorithm in an efficient manner. First, however, we analyze its approximation ratio for the 2PH and 1SH label models.

4.1.1 Approximation ratio for the 2PH and 1SH models

Lemma 4.1. *For the free-label-maximization problem with unit-square labels, the* GreedySweep *algorithm gives a* 6*-approximation for the 1SH model and a* 7*-approximation for the 2PH model, and both ratios are tight.*

Proof. Consider an optimal solution (for the given label model) and let $\textsc{opt}(p_i)$ denote the label assigned to point p_i in the optimal solution. Now suppose $\textsc{opt}(p_i)$ is free in the optimal solution. We will charge $\textsc{opt}(p_i)$ to a freeable label $\textsc{gs}(p_j)$ placed by GreedySweep, with $j \leqslant i$. Freeable labels remain free throughout the execution of GreedySweep, so this charges each free label in the optimal solution to a free label in the computed solution. We will first describe how the charging is done, and then argue that no label in the computed solution is charged more than a few times.

- Suppose $\textsc{opt}(p_i)$ is a freeable label candidate at the time p_i is being processed. Then $\textsc{gs}(p_i)$ is necessarily a freeable label—although possibly distinct from $\textsc{opt}(p_i)$—and we charge $\textsc{opt}(p_i)$ to $\textsc{gs}(p_i)$. We call $\textsc{opt}(p_i)$ a *type-0 charge* to $\textsc{gs}(p_i)$—see Figure 4.1(a).

- Suppose $\textsc{opt}(p_i)$ is *not* a freeable label candidate at the time p_i is being processed. We will now argue that there must still be a "nearby" freeable label to which we can charge.

  Consider an unprocessed point p_k—that is, a point p_k with $k > i$—and suppose we label it with its rightmost label candidate. Then this label cannot intersect $\textsc{opt}(p_i)$. Otherwise all other label candidates for p_k—which lie farther to the left—would intersect $\textsc{opt}(p_i)$ as well, contradicting that $\textsc{opt}(p_i)$ is free. In the same way, p_k's rightmost label candidate cannot intersect previously placed freeable labels. Hence the points still to be labeled cannot be the cause that $\textsc{opt}(p_i)$ is not freeable. Thus there must be a point p_j with $j < i$ which already has a label $\textsc{gs}(p_j)$ that intersects $\textsc{opt}(p_i)$.

  - Suppose $\textsc{gs}(p_j)$ is freeable. Then we charge $\textsc{opt}(p_i)$ to $\textsc{gs}(p_j)$ and call $\textsc{opt}(p_i)$ a *type-1 charge* to $\textsc{gs}(p_j)$—see Figure 4.1(b).
  - Suppose $\textsc{gs}(p_j)$ is *not* freeable. Note that $\textsc{gs}(p_j)$ cannot be the leftmost label candidate for p_j, otherwise all of p_j's label candidates intersect $\textsc{opt}(p_i)$. But if $\textsc{gs}(p_j)$ is not leftmost, then as we move $\textsc{gs}(p_j)$ to the left it must hit a freeable label before becoming leftmost. Let $\textsc{gs}(p_k)$ be the first freeable label hit, where $k < j$. Charge $\textsc{opt}(p_i)$ to $\textsc{gs}(p_k)$, and call $\textsc{opt}(p_i)$ a *type-2 charge* to $\textsc{gs}(p_k)$ through p_j—see Figure 4.1(c).

To finish the proof of the stated approximation ratios, we will now give tight upper bounds for the number of charges of each type to a freeable label.

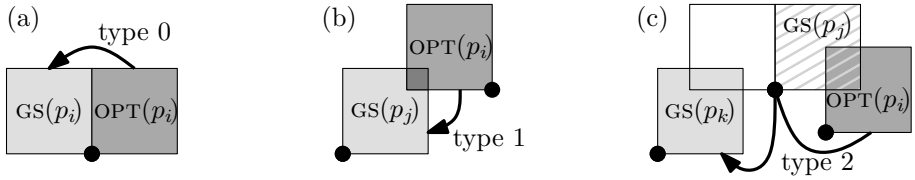


Figure 4.1: The three types of charges used in proving the approximation ratio of GreedySweep for the 2PH and 1SH models.

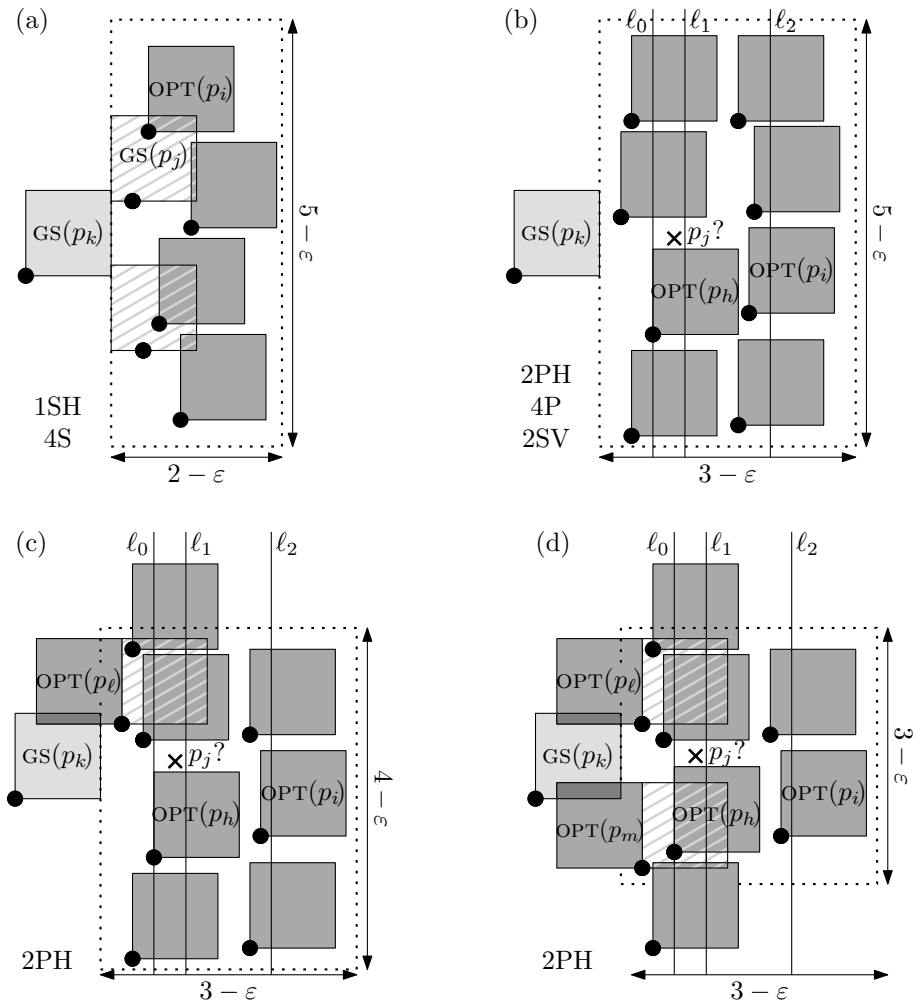


Figure 4.2: (a) In the 1SH model there can be four type-2 charges to a label. (b),(c),(d) When there are zero, one, or two type-1 charges, respectively, the total number of type-1 and type-2 charges is always at most seven.

- *Type 0.* Consider a label $\text{GS}(p_i)$ that receives a type-0 charge $\text{OPT}(p_i)$. Trivially, this must be the only type-0 charge to $\text{GS}(p_i)$. Note that $\text{GS}(p_i)$ must be at least as far to the left as $\text{OPT}(p_i)$, otherwise GreedySweep would have picked $\text{OPT}(p_i)$ over $\text{GS}(p_i)$. Thus any label for a point p_j to the right of p_i that intersects $\text{GS}(p_i)$ must also intersect $\text{OPT}(p_i)$, contradicting that $\text{OPT}(p_i)$ is free. This implies that $\text{OPT}(p_i)$ is not just the only type-0 charge to $\text{GS}(p_i)$, but the only charge of *any* type.

- *Type 1.* Consider a label $\text{GS}(p_j)$ that receives a type-1 charge $\text{OPT}(p_i)$. Note that $\text{OPT}(p_i)$ must contain a corner of $\text{GS}(p_j)$. If $\text{OPT}(p_i)$ contains the top-left or bottom-left corner of $\text{GS}(p_j)$ then all of p_i's label candidates intersect $\text{GS}(p_j)$. Thus $\text{OPT}(p_i)$ must contain the top-right or bottom-right corner of $\text{GS}(p_j)$. Hence there can be at most two type-1 charges to $\text{GS}(p_j)$.

- *Type 2.* Consider a label $\text{GS}(p_k)$ that receives a type-2 charge $\text{OPT}(p_i)$ through some point p_j. Note that $\text{GS}(p_j)$ will have been placed as far to the left as possible without intersecting freeable labels. Hence, in the 1SH model, $\text{GS}(p_j)$ touches $\text{GS}(p_k)$. In turn, $\text{OPT}(p_i)$ intersects $\text{GS}(p_j)$. Therefore $\text{OPT}(p_i)$ must be fully contained in a rectangle of size $(2-\varepsilon)\times(5-\varepsilon)$, for some $\varepsilon > 0$, whose left side contains the right side of $\text{GS}(p_k)$. In Figure 4.2(a) this rectangle is shown dotted. The optimal solution can have at most four free labels completely contained in this rectangle, so $\text{GS}(p_k)$ can be charged at most four times in this manner. Together with two type-1 charges, that means there is a total of at most six charges to $\text{GS}(p_k)$ in the 1SH model.

  In the 2PH model, $\text{GS}(p_j)$ in general does not touch $\text{GS}(p_k)$. Therefore the rectangle fully containing any possible $\text{OPT}(p_i)$ is of size $(3-\varepsilon)\times(5-\varepsilon)$, as seen in Figure 4.2(b). This would imply that $\text{GS}(p_k)$ can receive eight type-2 charges, giving a total of ten together with two type-1 charges. By examining the cases with zero, one, and two type-1 charges separately, we shall reduce this total to eight. Then, by analyzing the type-2 charges more carefully, we reduce this further to a bound of seven, which is tight in the worst case.

  If there are no type-1 charges to $\text{GS}(p_k)$, then obviously it receives at most eight charges in total, all of type 2. So suppose there is one type-1 charge to $\text{GS}(p_k)$, say $\text{OPT}(p_\ell)$, for some $\ell > k$. Then $\text{OPT}(p_\ell)$ must be leftmost and $\text{GS}(p_\ell)$ must be rightmost. There can be up to two type-2 charges to $\text{GS}(p_k)$ through p_ℓ: one intersecting the top-right corner of $\text{GS}(p_\ell)$ and one intersecting its bottom-right corner. All other type-2 charges must go through points distinct from p_ℓ. Say p_m is such a point. Then $\text{OPT}(p_m)$ is leftmost and intersects $\text{GS}(p_k)$ but not $\text{OPT}(p_\ell)$. Assume $\text{OPT}(p_\ell)$ contains the top-right corner of $\text{GS}(p_k)$ (the case where it contains the bottom-right corner is symmetric). Then $\text{OPT}(p_m)$ must be below $\text{OPT}(p_\ell)$. Further, $\text{GS}(p_m)$ has the same y-coordinate as $\text{OPT}(p_m)$, so all labels with a type-2 charge through p_m must be below the horizontal line through the top edge of $\text{OPT}(p_\ell)$. All type-2 charges to $\text{GS}(p_k)$ must therefore fit in a smaller $(3-\varepsilon)\times(4-\varepsilon)$ rectangle, except

possibly the topmost type-2 charge through p_ℓ—see Figure 4.2(c). At most six type-2 charges will fit in the rectangle, yielding a total of seven type-2 charges. Together with the type-1 charge $\textsc{opt}(p_\ell)$ that gives a total of eight charges in this case. Lastly, suppose there are two type-1 charges to $\textsc{gs}(p_k)$. Then we combine the arguments for the case where there is a single type-1 charge intersecting the top-right corner of $\textsc{gs}(p_k)$ and the case where there is a single type-1 charge intersecting the bottom-right corner of $\textsc{gs}(p_k)$. This gives that all type-2 charges must fit in a $(3-\varepsilon) \times (3-\varepsilon)$ rectangle, with the possible exception of two type-2 charges—see Figure 4.2(d). At most four type-2 charges will fit in the rectangle, for a total of six type-2 charges. Together with the two type-1 charges this again gives a total of eight charges.

To reduce the total from eight to seven, we first stab the dotted rectangles in Figure 4.2(b)–(d) with two vertical lines ℓ_1 and ℓ_2, with ℓ_1 at distance 1 from the left edge, and ℓ_2 at distance 1 from the right edge. Among the type-2 charges within the rectangle we call the ones intersecting ℓ_1 the *left column*, and the ones intersecting ℓ_2 the *right column*. Note that every type-2 charge in the rectangle must be in one of the columns, and no type-2 charge can be in both columns if we are to have eight charges in total. Let $\textsc{opt}(p_h)$ be the rightmost label in the left column, and note that there must be some label in the right column, say $\textsc{opt}(p_i)$, that lies completely to the right of $\textsc{opt}(p_h)$. For this label $\textsc{opt}(p_i)$ to be charged to $\textsc{gs}(p_k)$ there must be some point p_j with a leftmost label candidate intersecting $\textsc{gs}(p_k)$ and a rightmost label candidate intersecting $\textsc{opt}(p_i)$. This means that the horizontal distances between $\textsc{gs}(p_k)$ and p_j, and between p_j and $\textsc{opt}(p_i)$ must be less than 1. Thus p_j lies between the vertical line ℓ_1 and the vertical line ℓ_0 through the left edge of $\textsc{opt}(p_h)$. Furthermore, p_j can be neither above the top label in the left column, nor below the bottom label in the left column, as the vertical distance to $\textsc{gs}(p_k)$ is then greater than 1. Obviously p_j cannot be inside a free label, so it must be between two labels of the left column. However, the vertical distance between two such labels is necessarily less than 1, so one of them must be intersected by p_j's label. This contradicts that these labels were free in $\textsc{opt}$. Hence having eight charges is impossible, and there can be at most seven.

We conclude that the approximation ratios for the 1SH model and the 2PH model are at best 6 and 7, respectively. Figures 4.3 and 4.4 show that these ratios are tight. □

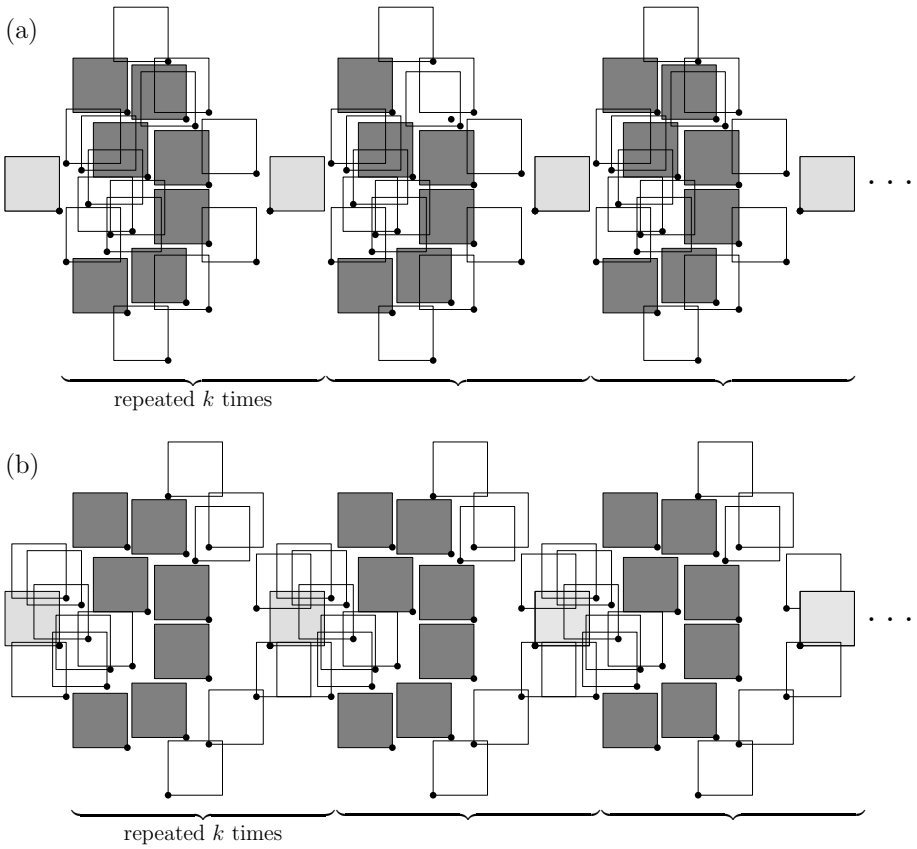


Figure 4.3: (a) A point set with its GreedySweep labeling for the 2PH model; the $k+1$ light gray labels are free. (b) A different solution for the same instance where the $7k$ dark gray labels are free.

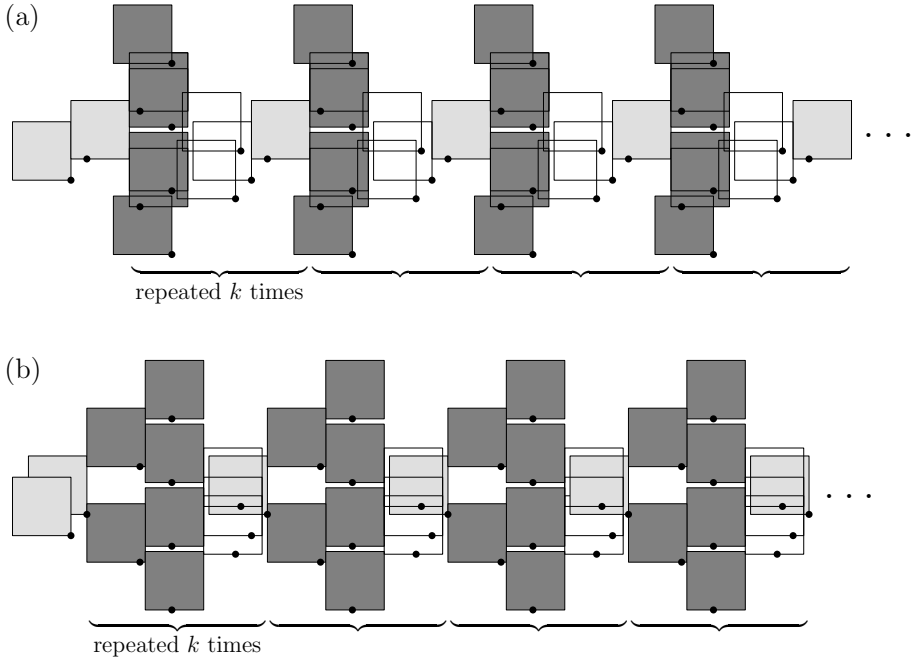


Figure 4.4: (a) A point set with its GreedySweep labeling for the 1SH model; the $k+2$ light gray labels are free. (b) A different solution for the same instance where the $6k$ dark gray labels are free.

4.1.2 Approximation ratio for the 4P, 2SV, and 4S models

For other label models, GreedySweep does not necessarily achieve a constant-factor approximation. We shall prove this for the 4P model in the following lemma.

Lemma 4.2. *For the 4P model, the approximation ratio of GreedySweep can be as bad as* $\Omega(\sqrt{n})$.

Proof. To prove this claim, we first divide the plane into identical, quadrilateral "tiles" by two sets of parallel lines. We "paint" each tile with the same pattern of one black and four white points. The shape of the tiles and the position of the points shown in Figure 4.5(a) is such that all label candidates contain points except the bottomleft label candidate for each black point. No white point can therefore have a free label. On the other hand, every white point *can* be labeled to avoid the bottomleft label candidates for the black points, so that those can indeed all be free in a solution. For some white points this requires using one

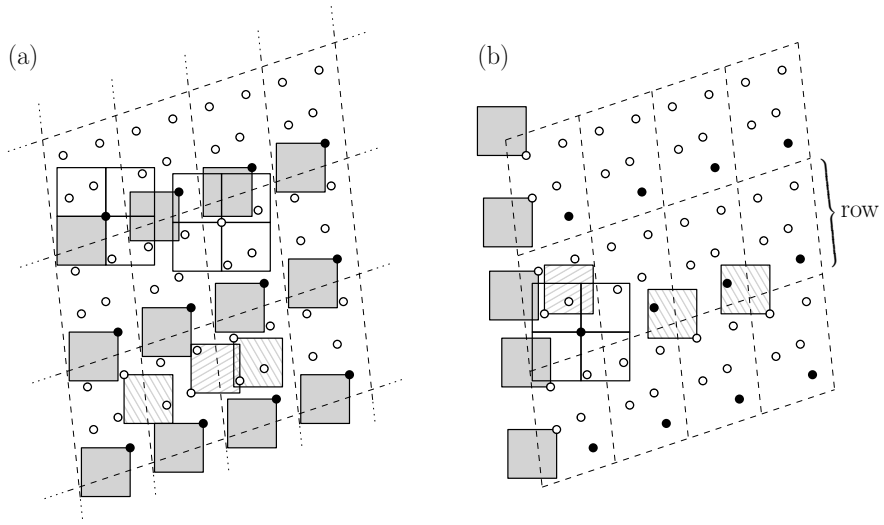


Figure 4.5: The approximation factor of GreedySweep can be $\Omega(\sqrt{n})$ for the 4P model.

of its two rightmost label candidates, however. This is key, as GreedySweep prefers leftmost label candidates, all other things being equal.

When we tile not the plane, but some finite area $\mathcal{A}$ in the above way, it becomes possible to assign free labels to some of the leftmost white points in each "row". GreedySweep will do so, thereby preventing the leftmost black point in the row from getting a free label, as in Figure 4.5(b). Its non-free label will have to be in a rightmost position, thereby blocking the next black point in the row from getting a free label. With no free labels "nearby", GreedySweep has no reason to prefer one non-free label candidate over another for the white points. It will then always pick one of the leftmost ones, say the topleft one, which blocks all remaining black points from getting free labels. The end result is that GreedySweep only has free labels along the perimeter of $\mathcal{A}$ while an optimal solution has a free label at every black point in the interior of $\mathcal{A}$. □

We can remedy the situation by running GreedySweep several times with different sweep directions and taking the best of the computed solutions.

- For the 2SV model we do one left-to-right sweep (as before) and one right-to-left sweep. In the latter sweep we modify the algorithm so that it prefers rightmost candidates rather than leftmost candidates.
- For the 2SH model we do one top-to-bottom sweep (preferring topmost label candidates) and one bottom-to-top sweep (preferring bottommost label candidates).

- For the 4P model we use the same two sweeps as for the 2SV model (but using the sweeps for the 2SH model would work as well).
- For the 4S models we sweep in all four directions.

Next we show that this always yields a constant-factor approximation.

Lemma 4.3. *The approximation factors of the algorithms described are* 22 *for the 4P and 2SV models, and* 32 *for the 4S model.*

Proof. Consider an optimal solution OPT and the solution GS computed by the left-to-right application of GreedySweep. Because of the above discussion we cannot hope to charge each free label in OPT to one in GS in a way that there are few charges to each label. However, we *can* charge the free *rightmost* labels in OPT in such a way. If most free labels in OPT turn out not to be rightmost (as above), then we can symmetrically consider the leftmost free labels in the right-to-left sweep, the bottommost free labels in the top-to-bottom sweep, or the topmost free labels in the bottom-to-top sweep. In this way, at least half the free labels in OPT can be charged for the 4P and 2SV models, and at least a quarter for the 4S model. The charging scheme is as before, but with one extra type of charge. Let p_i be a point with a free rightmost label in OPT.

- Suppose $\text{OPT}(p_i)$ is a freeable label candidate at the time p_i is being processed. We charge $\text{OPT}(p_i)$ to $\text{GS}(p_i)$ (which must also be freeable) and call $\text{OPT}(p_i)$ a *type-0 charge* to $\text{GS}(p_i)$—see Figure 4.6(a).
- Suppose $\text{OPT}(p_i)$ is a non-freeable label candidate (at the time p_i is being processed) because it is intersected by a label $\text{GS}(p_j)$ that has already been placed—that is, $j < i$.
  - If $\text{GS}(p_j)$ is freeable, then we charge $\text{OPT}(p_i)$ to $\text{GS}(p_j)$ and call $\text{OPT}(p_i)$ a *type-1 charge* to $\text{GS}(p_j)$—see Figure 4.6(b).
  - Suppose $\text{GS}(p_j)$ is *not* freeable. Note that since $\text{OPT}(p_i)$ is rightmost, and $j < i$, $\text{GS}(p_j)$ cannot be leftmost. But if $\text{GS}(p_j)$ is not leftmost, then as we move $\text{GS}(p_j)$ to the left it must hit a freeable label before it has become leftmost. Let $\text{GS}(p_k)$ be the first freeable label hit, where $k < j$. Charge $\text{OPT}(p_i)$ to $\text{GS}(p_k)$, and call $\text{OPT}(p_i)$ a *type-2 charge* to $\text{GS}(p_k)$ through p_j—see Figure 4.6(c).
- Suppose $\text{OPT}(p_i)$ is a non-freeable label candidate because it will inevitably be intersected by a label $\text{GS}(p_j)$ still to be placed—that is, $j > i$. (Note that this case could not occur in the 2PH and 1SH models.) This means that p_j has some label candidates that intersect $\text{OPT}(p_i)$, some label candidates that intersect one or more previously placed freeable labels, and no other label candidates. Since p_i and all points that have received a freeable label lie to the left of p_j, the same must be true of p_j's *rightmost* label

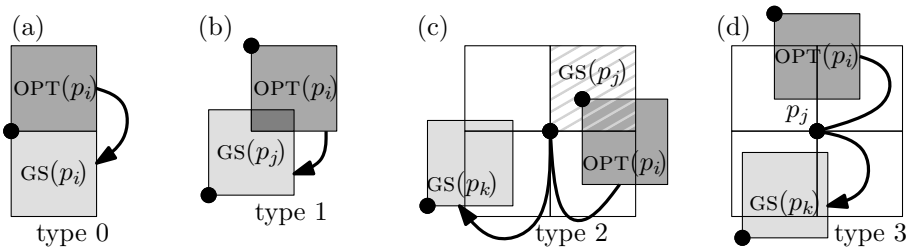


Figure 4.6: The four types of charges used in proving the approximation ratio of GreedySweep for the 4P, 2SV, and 4S models.

candidates. In other words, some of p_j's rightmost candidates intersect $\text{OPT}(p_i)$, and some intersect previously placed freeable labels. Let $\text{GS}(p_k)$ be such a previously placed freeable label. Charge $\text{OPT}(p_i)$ to $\text{GS}(p_k)$, and call $\text{OPT}(p_i)$ a *type-3 charge* to $\text{GS}(p_k)$ through p_j—see Figure 4.6(d).

To finish the proof of the stated approximation ratios, we will now give upper bounds for the number of charges of each type to a freeable label.

- *Type 0.* Consider a label $\text{GS}(p_i)$ that receives a type-0 charge $\text{OPT}(p_i)$. Trivially, this must be the only type-0 charge to $\text{GS}(p_i)$. If $\text{OPT}(p_i)$ and $\text{GS}(p_i)$ differ in y-coordinate, then there can still be additional charges of other types to $\text{GS}(p_i)$. However, $\text{OPT}(p_i)$ must contain (on its boundary if not in its interior) either the top-right or the bottom-right corner of $\text{GS}(p_i)$. That corner cannot be intersected by another free label of OPT, so by "adding" a type-0 charge to $\text{GS}(p_i)$ we "lose" one type-1 charge.

- *Type 1.* Consider a label $\text{GS}(p_j)$ that receives a type-1 charge $\text{OPT}(p_i)$. Note that $\text{OPT}(p_i)$ must contain a corner of $\text{GS}(p_j)$. Since $\text{OPT}(p_i)$ is rightmost, and $j < i$, $\text{OPT}(p_i)$ cannot contain the top-left or bottom-left corner of $\text{GS}(p_j)$. Thus $\text{OPT}(p_i)$ must contain the top-right or bottom-right corner of $\text{GS}(p_j)$, and hence there can be at most two type-1 charges to $\text{GS}(p_j)$.

- *Type 2.* Consider a label $\text{GS}(p_k)$ that receives a type-2 charge $\text{OPT}(p_i)$ through some point p_j. Note that $\text{GS}(p_j)$ will have been placed as far to the left as possible without intersecting freeable labels. Like for the 1SH model, this means that $\text{GS}(p_j)$ touches $\text{GS}(p_k)$ in the 4S model. In turn, $\text{OPT}(p_i)$ intersects $\text{GS}(p_j)$. Therefore $\text{OPT}(p_i)$ must be fully contained in a rectangle of size $(2-\varepsilon) \times (5-\varepsilon)$, for some $\varepsilon > 0$, whose left side contains the right side of $\text{GS}(p_k)$. This was illustrated for the 1SH model in Figure 4.2(a). The optimal solution can have at most four free labels completely contained in this rectangle, so $\text{GS}(p_k)$ can be charged at most four times in this manner.

In the 4P and 2SV models, as in the 2PH model, $\text{GS}(p_j)$ in general does not touch $\text{GS}(p_k)$. Therefore the rectangle fully containing $\text{OPT}(p_k)$ is of size $(3-\varepsilon)\times(5-\varepsilon)$, as was illustrated for the 2PH model in Figure 4.2(b). This would imply that $\text{GS}(p_k)$ can receive eight type-2 charges. We reduced this number to seven for the 2PH model, and the same argument applies for the 4P and 2SV models. We also argued for the 2PH model that the total number of type-1 and type-2 charges combined was no more than seven. It is not clear how to apply that reasoning to the 4P and 2SV models, however.

- *Type 3.* Consider a label $\text{GS}(p_k)$ that receives a type-3 charge $\text{OPT}(p_i)$ through some point p_j. Then some rightmost label candidates for p_j intersect $\text{OPT}(p_i)$, and some intersect $\text{GS}(p_k)$ or other previously placed freeable labels. In particular, consider the top-rightmost candidate $\text{NE}(p_j)$ and the bottom-rightmost candidate $\text{SE}(p_j)$. $\text{OPT}(p_i)$ cannot intersect both $\text{NE}(p_j)$ and $\text{SE}(p_j)$, for then $\text{OPT}(p_i)$ contains p_j. On the other hand, $\text{OPT}(p_i)$ must intersect one of $\text{NE}(p_j)$ and $\text{SE}(p_j)$. The same reasoning applies to $\text{GS}(p_k)$. So either $\text{OPT}(p_i)$ intersects $\text{NE}(p_j)$ and $\text{GS}(p_k)$ intersects $\text{SE}(p_j)$, or the other way around. In the former case $\text{OPT}(p_i)$ is above $\text{GS}(p_k)$, in the latter case $\text{OPT}(p_i)$ is below $\text{GS}(p_k)$. This leads to the two type-3 charges to $\text{GS}(p_k)$ depicted in Figure 4.7, and we claim that this is the maximum possible. To see this, consider the type-3 charge with the highest y-coordinate, say $\text{OPT}(p_i)$, and suppose it charges through the point p_j. Then other type-3 charges above $\text{GS}(p_k)$ with lower y-coordinates than $\text{OPT}(p_i)$ must contain p_j, a contradiction. A symmetric argument holds for the type-3 charges below $\text{GS}(p_k)$.

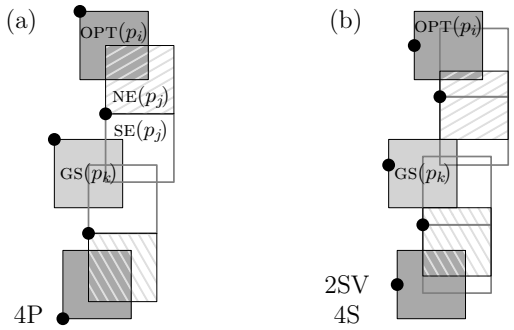


Figure 4.7: There can be at most two type-3 charges to a label in (a) the 4P model and (b) the 2SV and 4S models.

Summarizing, the maximum number of charges to a freeable label is eight for the 4S model (two of type 1, four of type 2, two of type 3) and eleven for the 4P and 2SV models (two of type 1, seven of type 2, two of type 3). Recall that we charge at least a quarter of the free labels of OPT in the 4S model, leading to an approximation ratio of $4 \cdot 8 = 32$. In the 4P and 2SV models we charge at least half the free labels of OPT, leading to an approximation ratio of $2 \cdot 11 = 22$.

We do not claim that these bounds are tight. Even if the bounds on the number of charges to a freeable label prove to be tight, that would not imply tightness of the approximation ratios. For example, a "bad" problem instance for the left-to-right sweep may turn out to be "good" for the right-to-left sweep. □

The following theorem states the main results of this section.

Theorem 4.1. *There are $O(n \log n)$-time and $O(n)$-space algorithms for free-label maximization on n points with unit-square labels, having the following approximation ratios:* 6 *(tight) for the 1SH model,* 7 *(tight) for the 2PH model,* 22 *for the 4P and 2SV models, and* 32 *for the 4S model.*

The approximation factors have been proved in Lemmas 4.1 and 4.3. In the next subsection we will describe how to implement GREEDYSWEEP to run in $O(n \log n)$ time and $O(n)$ space.

4.1.3 Efficient implementation

Recall our constraints on the labels that we place: 1) no label intersects a previously placed *freeable* label, 2) a freeable label does not intersect *any* previously placed label, and 3) every unprocessed point has at least one label candidate not intersecting a previously placed freeable label. For each of these conditions we will maintain a data structure that we can efficiently query for the set of label candidates satisfying the condition. More precisely, we will use:

1. A structure FREE storing the union of the *freeable* labels placed so far, supporting two operations:
   - ADDLABEL(FREE, L), which updates FREE for a newly placed freeable label L.
   - NONINTERSECTING(FREE, S), which returns, for a given set of label candidates S, the subset $S' \subseteq S$ of label candidates that do not intersect the labels in FREE.
2. An analogous structure ALL with operations ADDLABEL(ALL, L) and NONINTERSECTING(ALL, S) storing the union of *all* labels placed so far.
3. A structure R storing the "remaining" rightmost label candidates $\mathrm{R}(p) \subseteq \textsc{Cand}(p)$ for every unprocessed point p, that is, the rightmost label candidates not intersecting previously placed freeable labels. (Note that we

ignore any non-rightmost remaining label candidates. We will argue later that this is justified.) The structure supports the following three operations:

- UpdateForFreeLabel(R, L), which removes from R all label candidates intersecting the newly placed freeable label L.
- LeftmostNonBlocking(R, S), which returns the leftmost label candidates in S which do not "block" the remaining points from being labeled. A label candidate $L \in S$ blocks an unprocessed point p from being labeled if every remaining label candidate $L' \in \mathrm{R}(p)$ of p intersects L.
- RemoveCandidates(R, p), which removes all the label candidates for point p from R.

With these data structures, the GreedySweep algorithm becomes as follows. The temporary variables $\mathrm{LF}(p_i)$, $\mathrm{PF}(p_i)$, and $\mathrm{NF}(p_i)$, with $\mathrm{LF}(p_i) \subseteq \mathrm{PF}(p_i) \subseteq \mathrm{NF}(p_i) \subseteq \textsc{Cand}(p_i)$, are mnemonic and denote the leftmost freeable label candidates, the "potentially" freeable label candidates, and the non-freeable label candidates, respectively.

Algorithm GreedySweep(P, Cand)
1. Let $p_1, \ldots, p_n$ be the points in P, sorted in order of increasing x-coordinate.
2. Initialize the structures free, all, and R.
3. **for** $i \leftarrow 1$ **to** n
4. **do** RemoveCandidates(R, p_i) (* p_i can no longer be blocked *)
5. $\mathrm{PF}(p_i) \leftarrow$ NonIntersecting(all, Cand(p_i))
6. $\mathrm{LF}(p_i) \leftarrow$ LeftmostNonBlocking(R, $\mathrm{PF}(p_i)$)
7. **if** $\mathrm{LF}(p_i) \neq \emptyset$ (* p_i has a freeable label candidate *)
8. **then** Set L to be any label candidate in $\mathrm{LF}(p_i)$. (All lie equally far to the left.)
9. AddLabel(free, L)
10. UpdateForFreeLabel(R, L)
11. **else** $\mathrm{NF}(p_i) \leftarrow$ NonIntersecting(free, Cand(p_i))
12. Set L to be the leftmost label candidate in $\mathrm{NF}(p_i)$, with ties broken arbitrarily.
13. AddLabel(all, L)
14. Set gs(p_i) $\leftarrow L$.

It remains to demonstrate $O(n)$-space implementations of the proposed data structures with $O(\log n)$ amortized execution time for their operations.

Data structures for labels

To represent free and all it is not necessary to store the exact union of the labels in question; their right envelope will suffice. This is because points are

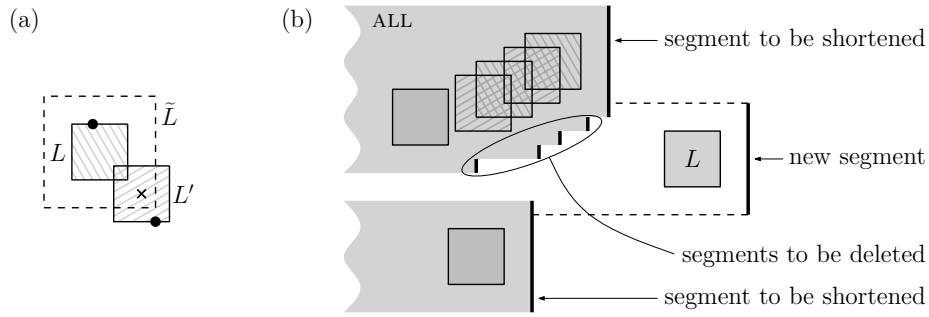


Figure 4.8: (a) L' intersects L if and only if its center is contained in the extended label $\widetilde{L}$. (b) The AddLabel operation inserts one new segment, shortens at most two existing segments, but may delete many segments.

processed from left to right, and labels touch the points to which they are attached. Such a right envelope can be maintained as a *red-black tree* [22, Ch. 13] storing its vertical segments. We order the tree nodes by y-coordinate, and *augment* [22, Ch. 14] each node v with a field storing the maximum x-coordinate in the subtree rooted at v. For easier queries we will not store the right envelope of the original labels, but of the *extended labels*. For a label L we define its extended label $\widetilde{L}$ to be the 2×2 square with the same center as L. A label L' then intersects L if and only if the center of L' lies inside $\widetilde{L}$—see Figure 4.8(a).

We now identify each label (candidate) with its center point. This turns $\textsc{Cand}(p_i)$ into a set of $O(1)$ points for the fixed-position models. For each such point p representing a label candidate, we can query the described red-black tree to determine in $O(\log n)$ time whether p lies to the right of the right envelope maintained in the tree. For the slider models this transformation turns $\textsc{Cand}(p_i)$ into a set of one, two, or four axis-parallel line segments. We can similarly query the red-black tree in $O(\log n)$ time with each such segment s to determine the part of s that lies to the right of the right envelope (if any). The result of each such query is either the empty set, a single point, or a (possibly shorter) line segment. Taking the union of these $O(1)$ query results completes the NonIntersecting$(\cdot\,, \textsc{Cand}(p_i))$ operation.

The AddLabel$(\cdot\,, L)$ operation inserts one new segment, and shortens at most two existing segments, but may delete many segments—see Figure 4.8(b). However, only $O(n)$ segment deletions will happen in total. Hence the operation can be performed in amortized $O(\log n)$ time, as required.

Data structures for unlabeled points

The structure R is used to maintain the invariant that every unlabeled point has label candidates available to it that do not intersect previously placed freeable

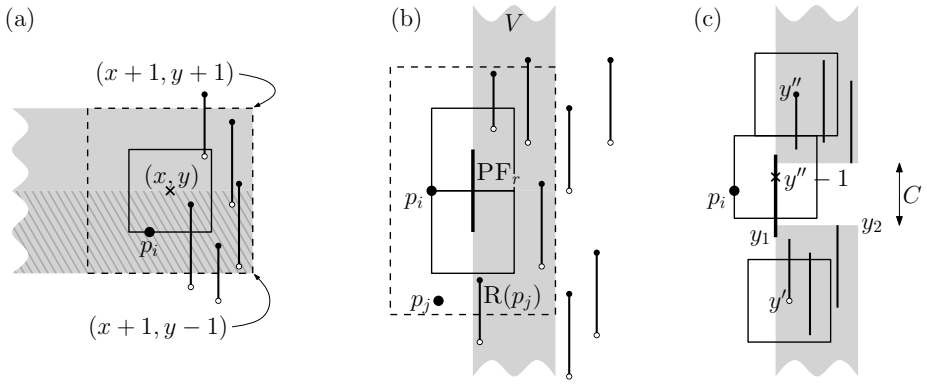


Figure 4.9: (a) $\mathrm{R}(p_j)$ is contained in $\widetilde{L}$ if and only if its bottommost point ($\circ$) is contained in the hatched area, or its topmost point ($\bullet$) in the unhatched area. (b) Only the segments in V can prevent label candidates of $\mathrm{PF}_r(p_i)$ from being freeable. (c) The freeable label candidates for p_i are in the interval $[y'+1, y''-1] \cap C$.

labels. Note that if an unprocessed point p_j has any such label candidate, it must have one that is also a rightmost label candidate for p_j. (We elaborated on this when discussing type-3 charges in the proof of Lemma 4.3.) Thus we defined $\mathrm{R}(p_j)$ as the rightmost label candidates of an unprocessed point p_j that do not intersect any previously placed freeable labels. Depending on the label model, $\mathrm{R}(p_j)$ is either a set of $O(1)$ points lying on a common vertical line, or a single vertical line segment.

LeftmostNonBlocking for fixed-position models. Determining the answer to the query LeftmostNonBlocking$(\mathrm{R}, \mathrm{PF}(p_i))$ boils down to computing for which label candidates $L \in \mathrm{PF}(p_i)$ there exists no unprocessed point p_j with $\mathrm{R}(p_j)$ fully contained in the extended label $\widetilde{L}$. To this end we maintain two *priority search trees* [10, Ch. 10] $\mathcal{P}_b$ and $\mathcal{P}_t$. In $\mathcal{P}_b$ we store the bottommost point of $\mathrm{R}(p_j)$ for all p_j, and in $\mathcal{P}_t$ the topmost point. For a single label candidate $L \in \mathrm{PF}(p_i)$ with center (x, y) we query $\mathcal{P}_b$ with the region $(-\infty, x+1) \times (y-1, y)$, and $\mathcal{P}_t$ with the region $(-\infty, x+1) \times (y, y+1)$. If the bottommost or topmost point of $\mathrm{R}(p_j)$ is found with these queries then $\widetilde{L}$ contains all of $\mathrm{R}(p_j)$, otherwise L is freeable—see Figure 4.9(a). This is because $\mathrm{R}(p_j)$ spans at most 1 unit of vertical space. Such queries can be performed in $O(\log n)$ time, so we can also execute LeftmostNonBlocking$(\mathrm{R}, \mathrm{PF}(p_i))$ in $O(\log n)$ time when $\mathrm{PF}(p_i)$ is a set of $O(1)$ points. That covers the fixed-position models.

LeftmostNonBlocking for slider models. For the slider models, let $\mathrm{PF}_\ell(p_i)$, $\mathrm{PF}_r(p_i)$, $\mathrm{PF}_t(p_i)$, and $\mathrm{PF}_b(p_i)$ denote the leftmost, rightmost, top-

most, and bottommost label candidates of $\mathrm{PF}(p_i)$, respectively. Since we identify a label candidate with its center point, each of these is either the empty set—so we can ignore it—, or an axis-parallel line segment. The vertical segment $\mathrm{PF}_\ell(p_i)$ is trivial to handle. It represents leftmost label candidates for the point p_i being processed, which can never intersect any of the rightmost label candidates $\mathrm{R}(p_j)$ for a point p_j still to be processed. So if $\mathrm{PF}_\ell(p_i) \neq \emptyset$, then $\mathrm{PF}_\ell(p_i)$ contains exactly the leftmost freeable label candidates, and we are done. Otherwise, we look at the horizontal segments $\mathrm{PF}_t(p_i)$ and $\mathrm{PF}_b(p_i)$. If the left endpoint of such a segment is not freeable, then no other point on the segment is either. Thus we simply query $\mathcal{P}_b$ and $\mathcal{P}_t$ with the left endpoints of $\mathrm{PF}_t(p_i)$ and $\mathrm{PF}_b(p_i)$ in the same way as above. If neither left endpoint represents a freeable label candidate we have to look at the vertical segment $\mathrm{PF}_r(p_i)$. This case is more complicated.

Let x be the x-coordinate of the vertical segment $\mathrm{PF}_r(p_i)$, and let V be the vertical strip between x-coordinates x and $x+1$. Note that only the vertical segments of R which lie inside V can cause points on $\mathrm{PF}_r(p_i)$ to be non-freeable—see Figure 4.9(b). We denote these segments by R_V. Within V we are essentially left with a 1-dimensional problem on intervals along the y-axis. We wish to determine the points y on interval $\mathrm{PF}_r(p_i)$ such that the open interval $(y-1, y+1)$ does not fully contain any interval of R_V. Note that such a point y cannot be contained in $\mathrm{R}(p_j)$ for any p_j, because $\mathrm{R}(p_j)$ has at most length 1 and would in turn be fully contained in $(y-1, y+1)$. Thus we need to look for such points y in the complement of the union of the intervals R_V. Consider one component C of the complement, and partition the intervals R_V into the set R_b below C and the set R_t above C. Let y' be the topmost bottom endpoint among the intervals in R_b, and let y'' be the bottommost top endpoint among the intervals in R_t. The interval $[y'+1, y''-1] \cap C$, if non empty, then consists of all freeable label candidates in C—see Figure 4.9(c). Our strategy is to loop over the components C overlapping $\mathrm{PF}_r(p_i)$, and determining the points y' and y'' for each of them.

In order to determine y' we build a red-black tree $\mathcal{T}_b$ on the y-coordinates of the bottom endpoints of the intervals in R_V. For determining y'' we build a similar tree $\mathcal{T}_t$ on the top endpoints. By performing searches in these two trees we can find y' and y'' in $O(\log n)$ time.

To find the components C, first note that there is at most one such component to be found. Otherwise there would be an interval $\mathrm{R}(p_j)$ fully contained in $\mathrm{PF}_r(p_i)$. The interval $\mathrm{R}(p_j)$ must have a length less than 1, so its top or bottom endpoint touches the right envelope maintained by FREE. Denoting the area to the left of this right envelope by $\mathcal{H}(\textsc{free})$, this means $\mathrm{PF}_r(p_i)$ is intersected by $\mathcal{H}(\textsc{free})$, and therefore by $\mathcal{H}(\textsc{all}) \supseteq \mathcal{H}(\textsc{free})$. However, that is impossible because $\mathrm{PF}(p_i) = \textsc{Cand}(p_i) \setminus \mathcal{H}(\textsc{all})$.

In order to find the component C we build an *interval tree* [10, Ch. 10] $\mathcal{I}$ on the intervals R_V. Let y_1 denote the bottom endpoint of $\mathrm{PF}_r(p_i)$. Using $\mathcal{I}$ we can determine in $O(\log n)$ time whether y_1 is contained in any interval of R_V.

If not, then $y_1 \in C$. Otherwise we can determine, in the same time bound, the maximum top endpoint y_2 among the intervals in which y_1 is contained. We then query $\mathcal{I}$ again to determine if y_2 is contained in any intervals. If not, we have $y_2 \in C$. Otherwise the intervals containing y_1 and the intervals containing y_2 together cover $\mathrm{PF}_r(p_i)$, so no suitable C exists.

Building the trees $\mathcal{T}_t$, $\mathcal{T}_b$, and $\mathcal{I}$ from scratch for each point would be too slow. However, as we process points from left to right the strip V moves monotonically from left to right. When a segment $\mathrm{R}(p_j)$ enters the strip we insert it into the three trees, and when it leaves the strip we remove it from the trees again, both in $O(\log n)$ time. As each point enters and leaves the strip at most once, this allows us to maintain these trees in amortized $O(\log n)$ time per point. This concludes the implementation details of the LeftmostNonBlocking operation.

Other operations on R. For the operation UpdateForFreeLabel(R, L) we first find all segments in R with an endpoint in $\widetilde{L}$. Using $\mathcal{P}_t$ and $\mathcal{P}_b$ this can be done in $O(\log n + k)$ time, where k is the number of segments found. We then cut off the parts of these segments in the interior of $\widetilde{L}$, and update all five aforementioned trees ($\mathcal{P}_t$, $\mathcal{P}_b$, $\mathcal{T}_t$, $\mathcal{T}_b$, and $\mathcal{I}$) for the shortened segments in $O(k \log n)$ time. In the worst case $k = \Omega(n)$. However, each segment can be shortened at most twice: once when its top endpoint falls within an extended freeable label, and once when its bottom endpoint does. It cannot be shortened more often as freeable labels do not intersect each other. Thus the amortized time needed for UpdateForFreeLabel(R, L) is $O(\log n)$.

The final operation, RemoveCandidates(R, p_i), simply removes $\mathrm{R}(p_i)$ from all five trees in $O(\log n)$ time. Thus we have shown an implementation of GreedySweep using $O(n \log n)$ time and $O(n)$ space, completing the proof of Theorem 4.1.

4.2 A PTAS for unit squares

We can obtain a PTAS for the case of unit-square labels by applying the "shifting technique" of Hochbaum and Maass [40]. Imagine a grid of unit squares overlaying the plane such that no point is on a grid line and call this the *1-grid*. If, for some integer $k > 4$ to be specified later, we leave out all but every k^{th} horizontal and vertical grid line this forms a coarser *k-grid*. By varying the offsets at which we start counting the lines, we can form k^2 different k-grids $G_1, \ldots, G_{k^2}$ out of the 1-grid. Consider one of them, say G_i. For any $k \times k$ square cell $\mathrm{C} \in G_i$, let $\overline{\mathrm{C}} \subset \mathrm{C}$ be the smaller $(k-4) \times (k-4)$ square with the same center as C—see Figure 4.10. We call $\overline{\mathrm{C}}$ the *inner cell* of C. For a given set P of n points, let $P_{\mathrm{C}} := \mathrm{C} \cap P$ and $P_{\overline{\mathrm{C}}} := \overline{\mathrm{C}} \cap P$. Furthermore, define $P_{\text{in}}(G_i) := \bigcup_{\mathrm{C} \in G_i} P_{\overline{\mathrm{C}}}$. We call a labeling $\mathcal{L}$ for P *inner-optimal* with respect to G_i if $\mathcal{L}$ maximizes the number of points in $P_{\text{in}}(G_i)$ that get a free label. Note that if $\mathrm{C}, \mathrm{C}' \in G_i$ are distinct cells, then a point $p \in \overline{\mathrm{C}}$ can never have a label intersecting the label

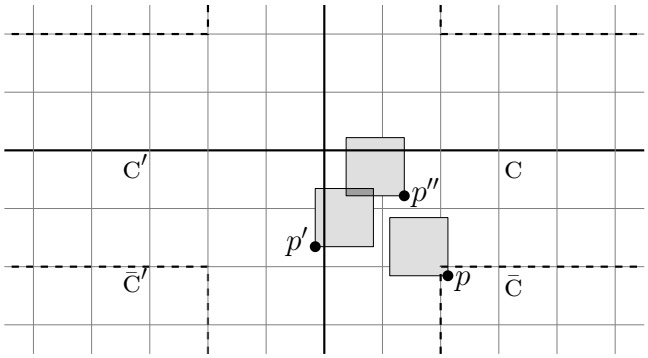


Figure 4.10: The cells (bold lines) and inner cells (dashed bold lines) of a k-grid. The underlying 1-grid is shown in thin gray lines.

for a point $p' \in \mathrm{C}'$—see Figure 4.10. Hence an inner-optimal labeling for P can be obtained by computing an inner-optimal labeling on P_{C} independently for each cell $\mathrm{C} \in G_i$. We will show below how to do this in time polynomial in n (but exponential in k). By itself this does not help us, as any particular k-grid G_i may have many points that lie outside of inner cells. We claim, however, that computing an inner-optimal labeling for all k-grids $G_1, \ldots, G_{k^2}$ and then taking the best one still yields a $(1+\varepsilon)$-approximation for suitably chosen k:

Lemma 4.4. *For all fixed-position and slider models, the best inner-optimal labeling for a set P of n points in the plane with respect to all k^2 different k-grids $G_1, \ldots, G_{k^2}$ yields a $(1+\varepsilon)$-approximation to free-label maximization with unit-square labels if $k \geqslant 6 + 8/\varepsilon$.*

Proof. Let OPT be some optimal solution, and let $F \subseteq P$ be the set of points with a free label in OPT. Let $f := |F|$. In any k-grid the inner cells are separated from each other by horizontal and vertical strips with a width of four 1-grid cells—see Figure 4.10. Thus any point in F lies in an inner cell for $(k-4)^2$ of the k^2 different k-grids. By the pigeon-hole principle, there must be a k-grid G_i for which $|F \cap P_{\mathrm{in}}(G_i)| \geqslant f \cdot (k-4)^2/k^2$. An inner-optimal labeling for P with respect to G_i will have at least $|F \cap P_{\mathrm{in}}(G_i)|$ free labels, hence we get a $(1+\varepsilon)$-approximation if $k^2/(k-4)^2 \leqslant 1+\varepsilon$. Solving for k yields

$$k \geqslant 4\,\frac{\sqrt{1+\varepsilon}}{\sqrt{1+\varepsilon}-1} = 4\,\frac{1+\varepsilon+\sqrt{1+\varepsilon}}{\varepsilon}.$$

Since $\sqrt{1+\varepsilon} \leqslant 1+\varepsilon/2$, the last term is upperbounded by $6+8/\varepsilon$. □

To complete the PTAS we need to show how to compute an inner-optimal labeling for the set P_{C} of points inside a $k \times k$ cell C. We say that a subset

$F \subseteq P_{\overline{C}}$ is *freeable* if we can label the points in P_C such that all points in F get a free label. The key insight is that, by a packing argument, not too many of the points $P_{\overline{C}}$ in the inner cell $\overline{C}$ can get a free label. Thus there is a limited number of freeable subsets. We first bound the number of potentially freeable subsets that we need to consider, and then show how to test each one for feasibility.

In many applications, there will not be too many points that are very close together (with respect to the label sizes). To take this into account, we will not just use the total number of points (n) in our analysis, but also their "density" (Δ). More precisely, we let $\Delta \leqslant n$ denote the maximum number of points in P contained in any unit square. If $\Delta = 1$ then labeling every point with its topleft label candidate, say, yields a solution where all labels are free. So we assume $\Delta \geqslant 2$ from now on.

Lemma 4.5. *Let C be a cell in a k-grid, let $\overline{C}$ be its inner cell, and let P_C and $P_{\overline{C}}$ be the respective subsets of points inside these cells. We can compute in $O(\sum_{F \in \mathcal{F}} |F|)$ time a collection $\mathcal{F}$ of subsets of $P_{\overline{C}}$ such that for any freeable subset $F \subseteq P_{\overline{C}}$ we have $F \in \mathcal{F}$. The collection $\mathcal{F}$ satisfies:*

- $|\mathcal{F}| \leqslant \Delta^{2(k-4)^2}$ *for the 2PH and 1SH models,*
- $|\mathcal{F}| \leqslant \Delta^{4(k-4)^2}$ *for the 4P, 2SV, and 4S models, and*
- $|F| \leqslant (k-3)^2$ *for all $F \in \mathcal{F}$.*

Proof. A 1-grid cell contains at most Δ points, and $\overline{C}$ consists of $(k-4)^2$ cells of the 1-grid. In the 2PH and 1SH models, no more than two points from the same 1-grid cell can be simultaneously labeled with non-intersecting labels. Thus any freeable subset $F \subseteq P_{\overline{C}}$ can be constructed by taking at most two points from each 1-grid cell. Hence, there are at most

$$\left(\binom{\Delta}{0} + \binom{\Delta}{1} + \binom{\Delta}{2}\right)^{(k-4)^2} \leqslant \Delta^{2(k-4)^2}$$

potentially freeable subsets F, where the inequality follows from the assumption that $\Delta \geqslant 2$. Similarly, no more than four points from the same 1-grid cell can be simultaneously labeled with non-intersecting labels in the 4P, 2SV, and 4S models, leading to at most

$$\left(\binom{\Delta}{0} + \binom{\Delta}{1} + \binom{\Delta}{2} + \binom{\Delta}{3} + \binom{\Delta}{4}\right)^{(k-4)^2} \leqslant \Delta^{4(k-4)^2}$$

potentially freeable subsets F.

In addition to limiting the number of points taken from each 1-grid cell, we can also limit the total number of points that we need to take. Since all labels are unit squares, the labels of the points in the $(k-4) \times (k-4)$ square $\overline{C}$ must be fully contained in a slightly larger $(k-2-\delta) \times (k-2-\delta)$ square around $\overline{C}$,

for some $\delta > 0$. There can be at most $(k-3)^2$ free labels in this area, hence $|F| \leqslant (k-3)^2$ for all freeable subsets $F \subseteq P_{\overline{C}}$. □

The previous lemma states that there are not too many potentially freeable subsets, and that each one is fairly small. The next two lemmas show, for different label models, how to test whether a potentially freeable subset is really freeable. We start with the 2PH, 1SH, and 4P models.

Lemma 4.6. *Let P_C be the set of all $n_C \leqslant \min(k^2\Delta, n)$ points contained in a k-grid cell C, and let $F \subseteq P_{\overline{C}}$ be a subset of those points. Let $f := |F|$. Then deciding whether there exists a labeling $\mathcal{L}$ for P_C where all points in F have a free label, and if so producing $\mathcal{L}$, can be done*

(i) in $O(n_C \log n_C)$ time for the 2PH and 1SH models, and

(ii) in $O((n_C - f)4^f)$ time for the 4P model.

Proof. (i) Go through the points from left to right and label them one-by-one. For every point $p \in F$ we pick the leftmost label candidate that does not intersect any previously placed label. For every point $p \in P_C \setminus F$ we pick the leftmost label candidate that does not intersect previously placed labels for points in F (but may intersect labels of other points). If we can process all points in P_C in this way then clearly we have found a suitable labeling $\mathcal{L}$. If we instead encounter a point p for which no label candidate can be chosen, then we report that no such labeling $\mathcal{L}$ exists. This is correct, because the partial labeling constructed by this algorithm has all labels at least as far to the left as $\mathcal{L}$ would have, so then the point p where we get stuck cannot be correctly labeled in $\mathcal{L}$ either. The above is simply a somewhat simplified version of the GreedySweep algorithm from Section 4.1, and can be implemented to run in $O(n_C \log n_C)$ time.

(ii) Enumerate all 4^f labelings of the points in F, and check for each such labeling $\mathcal{L}'$ whether it can be extended into a labeling $\mathcal{L}$ for all points in P_C. This entails checking whether each point $p \in P_C \setminus F$ has a label candidate that does not intersect any label of $\mathcal{L}'$. For this we only need to look at labels for points $p' \in F$ that lie in the 3×3 square of 1-grid cells centered at the 1-grid cell containing p. We can access these points efficiently by populating in advance a $(k-4) \times (k-4)$ array of "buckets" storing the points of F contained in each 1-grid cell. Since each 1-grid cell contains at most four points of F, performing the above check can then be done in $O(1)$ time for each of the $n_C - f$ points in $P_C \setminus F$. □

For the 2SV model we can neither use the greedy labeling (as for the 2PH and 1SH models), nor try all labelings of F (as for the 4P model). Instead we proceed as follows. Try all 2^f ways of restricting the labels for the points in F to be either leftmost or rightmost. The problem is then to decide whether F can be labeled with free labels in the 1SV model, while labeling $P_C \setminus F$ with (free and/or non-free) labels in the 2SV model. The position of a label along a 1-slider can

be modeled as a number between 0 and 1, making the configuration space $\mathcal{C}$ of possible labelings for F the f-dimensional unit hypercube. Let $\mathcal{C}_{\text{nonint}} \subseteq \mathcal{C}$ be the subspace of labelings for F where the labels of the points in F are disjoint. For any point $p \in P_{\text{C}} \setminus F$ let $\mathcal{C}_p \subseteq \mathcal{C}$ be the subspace of labelings $\mathcal{L}' \in \mathcal{C}_p$ for F where p can still get a label without intersecting a label in $\mathcal{L}'$. We then need to decide whether $\mathcal{C}_{\text{free}} := \mathcal{C}_{\text{nonint}} \cap \left(\bigcap_{p \in P_{\text{C}} \setminus F} \mathcal{C}_p\right)$ is non-empty, and if so construct a feasible labeling $\mathcal{L}' \in \mathcal{C}_{\text{free}}$ for F and extend it into a labeling $\mathcal{L}$ for P_{C}. We will show how this can be done using an arrangement of $O(n_{\text{C}})$ hyperplanes in $\mathcal{C}$.

Lemma 4.7. *Let P_C be the set of all $n_C \leqslant \min(k^2\Delta, n)$ points contained in a k-grid cell C, and let $F \subseteq P_C$ be a subset of those points. Let $f := |F|$. Then deciding whether there exists a labeling $\mathcal{L}$ for P_C where all points in F have a free label, and if so producing $\mathcal{L}$, can be done in $O(n_C)^f$ time for the 2SV and 4S models.*

Proof. We discuss only the 2SV model; the proof for the 4S model is analogous. For each point $p \in F$, we restrict its labeling to be either leftmost or rightmost—this can be done in 2^f ways—and from now on we consider this restriction to be fixed.

If two points $q, q' \in F$ have intersecting labels, then they must be fairly close. Specifically, there is a 4×5 rectangle $\mathcal{B}_q$ around q, consisting of 20 cells of the 1-grid, such that $\mathcal{B}_q$ contains q'—see Figure 4.11(a). Assuming it is possible for q and q' to have intersecting labels in the first place, preventing that from happening introduces a linear constraint on the slider coordinates of q and q'. Since F has at most four points in any 1-grid cell, $\mathcal{B}_q$ contains at most 80 points of F (including q itself). Hence $\mathcal{C}_{\text{nonint}}$ is the intersection of at most $f \cdot (80-1)/2 = 79f/2$ half-spaces. (The division by 2 is because otherwise the half-space for each pair q, q' would be counted twice, once for q and once for q'.)

For any point $p \in P_{\text{C}} \setminus F$, let $\mathcal{C}_p^\ell \subseteq \mathcal{C}$ be the subspace of labelings for F which still allow p to get a leftmost label. Now consider a labeling for F that is *not* in $\mathcal{C}_p^\ell$. Any leftmost label for p will intersect the label of at least one point of F in such a labeling. We claim (and will argue later) that there must then exist a subset $F' \subseteq F$ with $|F'| \leqslant 2$ such that every leftmost label of p intersects a label of a point in F'. Hence, $\mathcal{C}_p^\ell$ can be constructed as $\mathcal{C}_p^\ell = \bigcap_{F' \subseteq F,\, |F'| \leqslant 2} \mathcal{C}_p^\ell(F')$, where $\mathcal{C}_p^\ell(F')$ is the subspace of labelings for F where p has at least one leftmost label candidate not intersecting the labels for F'. Since $F' \subseteq \mathcal{B}_p$, there are at most $\binom{80}{1} + \binom{80}{2} = 3240$ sets F' to be considered. For $q \in F$ the subspace $\mathcal{C}_p^\ell(\{q\})$ takes on one of two shapes:

- If no label candidate of q by itself intersects all leftmost label candidates of p, then $\mathcal{C}_p^\ell(\{q\})$ is the full hypercube.
- Otherwise, $\mathcal{C}_p^\ell(\{q\})$ is a half-space defined by a linear constraint on q's slider coordinate, giving it a minimum or maximum value—see Figure 4.11(b).

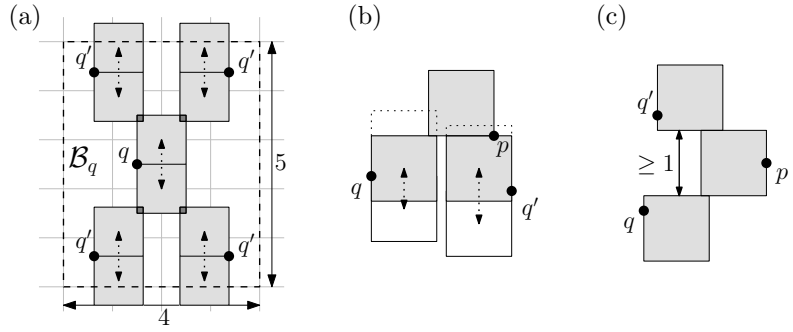


Figure 4.11: (a) Any point q' whose label intersects the label of a point q must lie in a region $\mathcal{B}_q$ around q, consisting of 20 cells of the 1-grid. (b) The labels of points $q, q' \in F$ on one side of p may not cross the horizontal line through p. (c) The labels of points $q, q' \in F$ on opposite sides of p need to be at least 1 unit apart vertically.

For $q, q' \in F$ the subspace $\mathcal{C}_p^\ell(\{q, q'\})$ takes on one of three shapes:

- If there is no pair of label candidates for q and q' that together intersect all leftmost label candidates of p, then $\mathcal{C}_p^\ell(\{q, q'\})$ is the full hypercube.
- Otherwise, suppose p is vertically in between q and q' (for example, q is below p and p is below q'). Then $\mathcal{C}_p^\ell(\{q, q'\})$ is a half-space defined by the linear constraint that the vertical distance between the labels of q and q' should be at least 1—see Figure 4.11(c).
- Lastly, if q and q' are on the same side (vertically) of p, then $\mathcal{C}_p^\ell(\{q, q'\}) = \mathcal{C}_p^\ell(\{q\}) \cap \mathcal{C}_p^\ell(\{q'\})$—see Figure 4.11(b).

Hence, $\mathcal{C}_p^\ell$ can be constructed as the intersection of at most 3240 half-spaces. The same is true for $\mathcal{C}_p^r$, the subspace of labelings for F which allow p to get a rightmost label. Since $\mathcal{C}_p = \mathcal{C}_p^\ell \cup \mathcal{C}_p^r$, we can find $\mathcal{C}_{\text{free}} = \mathcal{C}_{\text{nonint}} \cap \left(\bigcap_{p \in P_C \setminus F} \mathcal{C}_p\right)$ as the union of some of the cells in an arrangement of (at most) $h := 79f/2 + 2 \cdot 3240 n_C = O(n_C)$ hyperplanes. We can construct this arrangement in $O(h^f)$ time [29]. In the same time we can test whether $\mathcal{C}_{\text{free}}$ is non-empty and if so construct a labeling $\mathcal{L}' \in \mathcal{C}_{\text{free}}$ for F. Greedily extending the labeling $\mathcal{L}'$ for F into a labeling $\mathcal{L}$ for $P_C \supseteq F$ does not increase the running time.

Recall that we have to do all this for each of the 2^f ways of restricting the labeling of the points in F to leftmost or rightmost. Hence, the overall running time is $2^f \cdot O(n_C)^f = O(n_C)^f$.

It remains to prove the claim that we can ignore sets $F' \subseteq F$ with three or more elements. To this end, consider a labeling $\mathcal{L}'$ for F which intersects all leftmost label candidates for p. Let L be the topmost label in $\mathcal{L}'$ that intersects

p's bottom-leftmost label candidate, and let L' be the bottommost label in $\mathcal{L}'$ that intersects p's top-leftmost label candidate (possibly with $L' = L$). Then L and L' together must intersect all leftmost label candidates for p, otherwise p would have a free leftmost label candidate vertically between L and L'. □

Putting together the above lemmas yields the following result.

Theorem 4.2. *For any fixed $\varepsilon > 0$, and for each of the fixed-position and slider models, there exists a polynomial-time algorithm that computes a $(1+\varepsilon)$-approximation for free-label maximization with unit-square labels.*

The worst-case running time of the algorithm is $n^{O(1/\varepsilon^2)}$. If the density Δ of the point set is $O(1)$—that is, any unit square contains $O(1)$ points—then the running time improves to $O(n \log n) + n2^{O(1/\varepsilon^2)}$ for the 2PH, 4P, and 1SH models, and to $O(n \log n) + n2^{O(1/\varepsilon^2 \log(1/\varepsilon))}$ for the 2SV and 4S models.

Proof. Compute a 1-grid in $O(n \log n)$ time [40]. Let $k = \lceil 6+8/\varepsilon \rceil$ and generate all k^2 possible k-grids $G_1, \ldots, G_{k^2}$ out of the 1-grid. For each k-grid G_i, we compute an inner-optimal labeling for the (at most n) cells containing points. This is done for a cell $\mathrm{C} \in G_i$ by enumerating the potentially freeable subsets F of $P_{\overline{\mathrm{C}}}$ (Lemma 4.5), and checking for each subset F whether P_{C} can be labeled so that all points in F have a free label (Lemmas 4.6 and 4.7). The best out of these k^2 solutions is a $(1+\varepsilon)$-approximation (Lemma 4.4). Let $n_{\mathrm{C}} \leqslant \min(\Delta k^2, n)$ be the maximum number of points in any k-grid cell. The running time is then

$$O(n \log n) + nk^2 \cdot \begin{cases} \Delta^{2(k-4)^2} \cdot O\big(n_{\mathrm{C}} \log(n_{\mathrm{C}})\big) & \text{for the 2PH and 1SH models,} \\ \Delta^{4(k-4)^2} \cdot O\big(n_{\mathrm{C}}\, 4^{(k-3)^2}\big) & \text{for the 4P model,} \\ \Delta^{4(k-4)^2} \cdot O\big(n_{\mathrm{C}}\big)^{(k-3)^2} & \text{for the 2SV and 4S models.} \end{cases}$$

For $\Delta = O(n)$ all these running times are $\lceil 6 + 8/\varepsilon \rceil^2 n^{O(1/\varepsilon^2)}$, which simplifies to $n^{O(1/\varepsilon^2)}$ (assuming $n > 1$). For $\Delta = O(1)$ this improves to $O(n \log n) + n2^{O(1/\varepsilon^2)}$ for the 2PH, 4P, and 1SH models, and to $O(n \log n) + n2^{O(1/\varepsilon^2 \log(1/\varepsilon))}$ for the 2SV and 4S models. □

4.3 Discussion and future research

Inspired by air-traffic control and other applications where moving points need to be labeled, we have introduced *free-label maximization* as a new variant of the labeling problem, and have studied it for static points as a first step. In free-label maximization we must label all points with labels of fixed dimensions and seek to maximize the number of *free* labels (labels that do not intersect other labels). We have presented a simple and efficient constant-factor approximation algorithm, as well as a PTAS, for free-label maximization under the commonly assumed model that labels are directly attached to their points. In air-traffic

control, however, labels are usually connected to their point by means of a short line segment (a *leader*). This presents an interesting direction for future research. Our constant-factor approximation can definitely be extended to this case, but the approximation factors will be considerably worse and dependent on the length of the leaders. We believe the PTAS may be extendable as well, although this involves more technical details.

Our current algorithms work if all labels are unit squares (or, equivalently, all labels are translates of a fixed rectangle). The cases of labels being unit-height rectangles or arbitrary rectangles are still open. For the number-maximization problem these cases allow, respectively, a PTAS [18] and an $O(\log \log n)$-approximation [17]. The former achieves a $(1 + 1/k)$-approximation to number maximization in only $O(n \log n + n\Delta^{k-1})$ time, while the running time of our PTAS for free-label maximization is completely impractical. It would be interesting to see if these results for number maximization can be matched for free-label maximization. If not, then free-label maximization is strictly harder to approximate than number maximization, while easier than size maximization. The weighted version of the free-label-maximization problem is another interesting direction for future research.

The most important area for future research, however, is the labeling of moving points. Even outside of air-traffic control applications, we believe that free-label maximization can be a better model for this than the size- and number-maximization problems. Continuously scaling labels under size maximization would be hard to read, and the (dis)appearance of a label under number maximization is an inherently discrete event which can be disturbing for the viewer. It would be relatively simple to develop a *kinetic data structure* [6] to compute how our GreedySweep labeling changes over time as the points move. This is not enough to obtain a good result, however, as there will be times where the labeling undergoes discrete changes, with labels "jumping" from place to place. In the next chapter we take a different approach that does result in smooth label movements. While it gives no guaranteed approximation ratio on the number of free labels, it performs well in experiments. Finding a dynamic labeling method that also performs well in theory remains an elusive holy grail.

A Heuristic Algorithm for Dynamic Point Labeling

5

In Chapter 3 we proved the PSPACE-hardness of computing optimal dynamic point labelings. We then took a detour in Chapter 4 to develop an algorithm for computing static labelings that (approximately) maximize the number of free labels, with the promise that this would help us develop a heuristic algorithm for dynamic labeling. In this chapter we make good on that promise, by exhibiting such a heuristic algorithm and its experimental evaluation. We start by defining what qualities the computed dynamic labeling should ideally possess.

5.1 Problem definition

Recall that we defined a *dynamic point set* P as specifying for each point $p \in P$ the time at which it is added to the point set, the time at which it is removed from the point set, and its continuous trajectory. We refer to the time interval during which p is present in the point set as its *lifespan* and denote it by $\mathit{life}(p) = [\mathit{birth}(p), \mathit{death}(p)]$, where $\mathit{birth}(p)$ is the time at which p is added and $\mathit{death}(p)$ is the time at which it is removed. Whenever $t \in \mathit{life}(p)$ we say that p is *alive* at time t, and then denote its position by $p(t)$. In other words, we use "p" to refer both to a moving point and its trajectory, letting context determine which of the two is meant. For simplicity, we will assume throughout this chapter that the trajectories of the points are piecewise linear. Our results can also be extended to curved trajectories, however, as will be discussed at the end of the chapter.

5.1.1 Labeling desiderata

For a dynamic point set P we then seek a *dynamic labeling* $\mathcal{L}$, which for all t assigns a static labeling $\mathcal{L}(t)$ to the point set $P(t)$. Here $P(t) = \{p(t) \mid p \in P, t \in \mathit{life}(p)\}$ denotes the set of points that are alive at time t, at their respective locations. For such a dynamic labeling to be considered of high quality, we

naturally require each of these static labelings to be of high quality. That is, for all t we require the following of $\mathcal{L}(t)$:

(1) The association between points and labels should be clear.

(2) As many labels as possible should be as readable as possible.

Unique to dynamic labelings, we will additionally require of $\mathcal{L}$ that:

(3) The labels should not obscure the direction of movement of the points.

(4) The labels should move slowly and continuously, where we measure the speed of a label relative to the point it labels. (If $p \in P$ moves quickly then p's label may also move quickly, but it should move slowly in the moving coordinate system with p at the origin.)

Without these latter two conditions one could simply use any fast static labeling algorithm to relabel the point set from scratch every time the display is refreshed. The result would be very unpleasant and confusing to look at, however, with erratically moving labels distracting the user and obscuring the movement pattern of the points.

We must formalize these four desiderata before we can compute dynamic labelings that achieve them. We choose to use a 4-slider label model. This implies that labels are directly attached to their points, giving us condition (1) for free. An alternative would be to allow some distance between a point and its label and draw a short line segment between the two, called a *leader*. This is conventional in air-traffic control [25], where moving points represent airplanes. To achieve a clear association between points and labels would then require the leaders to only cross at large angles [41]. We have left exploring this option to future research.

Condition (2) was dealt with extensively in the introduction. For readability, labels should not overlap and be of sufficient size. This has led to research into three optimization problems for static labeling. In *size maximization* all points are to be labeled, and we seek the maximum label size at which this is possible without overlap. In *number maximization* a maximum-size subset of the points is to be labeled without overlap, using labels of fixed dimensions. In *free-label maximization*, lastly, all points are to be labeled with labels of fixed dimensions, and we wish to maximize the number of labels without overlap. Of these three, free-label maximization seems most suited for smooth dynamic labelings. The (dis)appearance of a label in number-maximization is an inherently discrete event, and continuously scaling labels would be hard to read. It remains, then, to choose a proper generalization of free-label maximization for dynamic labeling. We may wish to maximize, for example, the integral of the number of free labels over time, or the minimum percentage of free labels at any point in time.

Condition (3) is for the movement direction of the points to remain clear once their labels are added. To this end, we require the labels to drag "behind"

Figure 5.1: The α-trailing model for moving points (depicted on the right) is a subset of the 4-slider model (depicted on the left) where the label stays behind its point.

their points. More precisely, at each point in time we draw two rays emanating from the point, one in the point's direction of movement, and the other through the center of its label. We then require that these rays make an angle of at least α, as depicted in Figure 5.1. Here $\alpha \in [0, \pi]$ is a given parameter that specifies how strict we are in our requirement of behindness. In other words, the label model we use is not the 4-slider model but a subset of it that we call the *α-trailing model*.

Condition (4), lastly, requires us to move the labels continuously over time, and as slowly as possible (relative to the points being labeled). This leaves open whether we should minimize the maximum label speed or the average label speed. As it turns out, we can do both simultaneously (under some conditions), as will be shown in in Section 5.2.1. Thus we do not have to choose one or the other.

5.1.2 Interaction of label speed, behindness, and freeness

Instead of specifying the behindness parameter α and minimizing label speed, one could also consider specifying a maximum label speed and minimizing α. We shall not do so, however, as this would give the labeling algorithm no incentive to not use the maximum allowed speed whenever labels need to move. Moreover, a limit on the label speed may severely affect the number of free labels, as expressed by the following theorem.

Theorem 5.1. *Suppose we want to label a set of n moving points using a 1-slider or α-trailing model. If we then impose a maximum label speed, the best possible dynamic labeling can be substantially worse than if we did not limit label speed: the integral of free labels over time can be $\Theta(n)$ times worse, and the minimum number of free labels at any time can go from non-zero to zero.*

Proof. Let $k = n/2$, and assume for simplicity that it is an integer. We now construct two sets of k moving points each, $P = \{p_1, \ldots, p_k\}$ and $Q = \{q_1, \ldots, q_k\}$, as illustrated in Figure 5.2. All points move from left to right along straight-line trajectories, with unit-height labels trailing behind them on the left. The

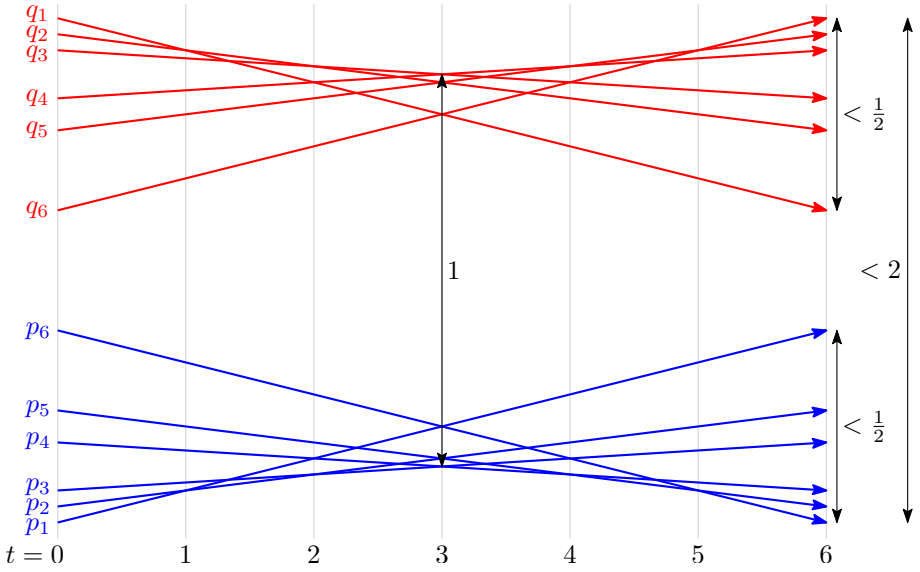


Figure 5.2: Consider a set of points moving along the depicted trajectories in such a way that they lie on a common vertical line which moves to the right at unit speed. In other words, $x\big(p_i(t)\big) = x\big(q_i(t)\big) = t$ for all i and t. Then the best speed-restricted dynamic labeling with trailing labels can be substantially worse than a dynamic labeling without restrictions on label speed.

speeds of the points are chosen such that all points lie on a common vertical line that moves at unit speed in the x-direction. The slopes of the trajectories are chosen so that during the time interval $[i-1, i]$ the lowest point is p_i and the highest point is q_i, for all $i \in \{1, \ldots, k\}$. Furthermore, the distance between the lowest and the highest point always lies in the range $[1, 2)$ during the time interval $[0, k]$. Lastly, the distance between any two points in P, and between any two points in Q, is less than $1/2$ throughout $[0, k]$.

This construction ensures that in a static labeling of $(P \cup Q)(t)$ for any time $t \in [0, k]$ there can be two free labels, but no more. The lowest point will then have a free label placed below it, the highest point will have a free label placed above it, and all other labels are placed to overlap each other in between the two. Now consider any dynamic labeling $\mathcal{L}$ of $P \cup Q$ which is restricted to have a maximum label speed of $1/(2k)$. Suppose $\mathcal{L}$ assigns the point p_i a free label somewhere in the interval $[i-1, i]$ where it is the lowest point. This then makes it impossible for any point p_j with $j \neq i$ to ever have a free label, as there is simply not enough time to move the label of p_i out of the way. At a speed of $1/(2k)$ the label can move by at most $1/2$ in the interval $[0, k]$, but the distance between p_i and p_j is always less than $1/2$. Hence, during the interval $[j-1, j]$

where p_j is the lowest point, p_i's label must overlap p_j's. A similar argument applies to Q and its highest point.

In conclusion, the integral over time of the number of free labels can be at most 2 in any dynamic labeling of $P \cup Q$ with a maximum label speed of $1/(2k)$. This is achieved by assigning a free label to p_i during $[i-1, i]$ and to q_j during $[j-1, j]$, for a single i and a single j, while having no free labels outside these intervals. In contrast, with a maximum label speed of $1/\varepsilon$, a free label can be assigned to p_i during $[i-1, i-\varepsilon]$ and to q_i during $[i-1+\varepsilon, i]$ for all i. There is then at least 1 free label at any time, and the number of free labels integrated over time is $2k(1-\varepsilon)$. Thus, when imposing a label speed limit of $1/(2k)$, the integral of free labels goes down from $2k(1-\varepsilon)$ to 2, while the minimum number of free labels at any time goes down from 1 to 0. By scaling the trajectories appropriately, the same can be achieved for any other given label speed limit. □

5.2 A heuristic algorithm

We propose a dynamic labeling algorithm that computes a series of feasible *static* labelings at certain moments in time, and then moves the labels smoothly from their positions in one static labeling to their positions in the next. In computing the static labelings we try to maximize the number of free labels, in computing the interpolation between labelings we try to minimize the movement speed of the labels. Thus we hope to achieve a good dynamic labeling for all four quality criteria mentioned in Section 5.1.1. The algorithm can be expressed by the following pseudocode, where $\Delta t > 0$ is the time between successive static labelings, and $\mathcal{L}(t, t')$ refers to the part of a dynamic labeling $\mathcal{L}$ between times t and t'.

Algorithm InterpolativeLabeling(P, $t_{\max}$)
1. $\mathcal{L} \leftarrow \emptyset$; $t \leftarrow 0$
2. $\mathcal{L}(t) \leftarrow$ StaticLabeling(P, t)
3. **while** $t < t_{\max}$
4. **do** $t_{\text{next}} \leftarrow \min(t + \Delta t, t_{\max})$
5. $\mathcal{L}(t_{\text{next}}) \leftarrow$ StaticLabeling(P, t_{next})
6. $\mathcal{L}(t, t_{\text{next}}) \leftarrow$ Interpolate$(\mathcal{L}(t), t,\ \mathcal{L}(t_{\text{next}}), t_{\text{next}})$
7. $t \leftarrow t_{\text{next}}$
8. **return** $\mathcal{L}$

This method has three parameters:

- The size of the timestep Δt. Smaller timesteps may lead to a greater number of free labels over time. Larger timesteps may lead to less or slower label movement, and fewer calls to the StaticLabeling subroutine.

- The algorithm used for StaticLabeling. We use the FourGreedySweeps algorithm developed in Chapter 4, which yields a constant-factor approximation to free-label maximization of unit-square labels in the 4-slider model. While the technique used to prove this approximation ratio does not carry over to the α-trailing model, the algorithm itself does.

- The algorithm used for Interpolate. We shall use a simple, linear-time algorithm based on computing shortest paths. It minimizes both the average and the maximum movement speed of each label. This algorithm is described next.

5.2.1 Interpolation algorithm

We are given two moments in time t_1 and t_2, and static labelings $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ of the dynamic point set at those times. We then wish to compute a dynamic labeling $\mathcal{L}(t_1, t_2)$ which *respects* them, that is, whose static labelings at times t_1 and t_2 equal $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$. We will do this independently for each moving point $p \in P$ with $\mathit{life}(p) \cap [t_1, t_2] \neq \emptyset$. For the rest of this section, let p be such a point and let $[t'_1, t'_2] = \mathit{life}(p) \cap [t_1, t_2]$.

Recall that the position of a label is restricted in that it must contain the point it labels on its boundary. Furthermore, the α-trailing label model specifies that the label's center lies "behind" the point. Specifically, a ray from the point in its direction of movement must make an angle of at least α with a ray from the point through its label's center. Now suppose we translate the coordinate system so that point p is always at the origin. The allowed positions for the center of p's label then trace out part of the surface of a box in 3-dimensional space-time. We refer to this *configuration space* for point p over the interval $[t'_1, t'_2]$ as $\mathcal{C}$. For $\alpha = \pi/2$, Figure 5.3(b) shows an example of such a configuration space for the point trajectory shown in Figure 5.3(a).

A label trajectory for p corresponds to a path through $\mathcal{C}$, monotone with respect to the time axis. To compute one, it will be convenient to "unfold" $\mathcal{C}$ into a rectilinear polygon R as in Figure 5.3(c). If point trajectory p has k vertices, then R is the union of $k-1$ closed, axis-aligned rectangles. The intersection of any two consecutive rectangles is a vertical line segment which we refer to as a *portal*—see Figure 5.3(c). If a portal's coordinate along the time axis is t, we refer to it as the *portal at* t, denoted by $\Psi(t)$. In addition to these $k-2$ portals we define two extra portals at t'_1 and t'_2. These are simply the sets of label positions that respect $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$. If $t'_1 = t_1$ then $\Psi(t'_1)$ is a single point representing the label given to p by $\mathcal{L}(t_1)$. If $t'_1 \neq t_1$ then $\mathcal{L}(t_1)$ specifies nothing about p's label position at time t'_1, and $\Psi(t'_1)$ is simply the leftmost edge of R. Analogously, $\Psi(t'_2)$ is either the rightmost edge of R, or a point representing p's label in $\mathcal{L}(t_2)$. With these definitions, a label trajectory for p corresponds to a time-monotone path through R from portal $\Psi(t'_1)$ to

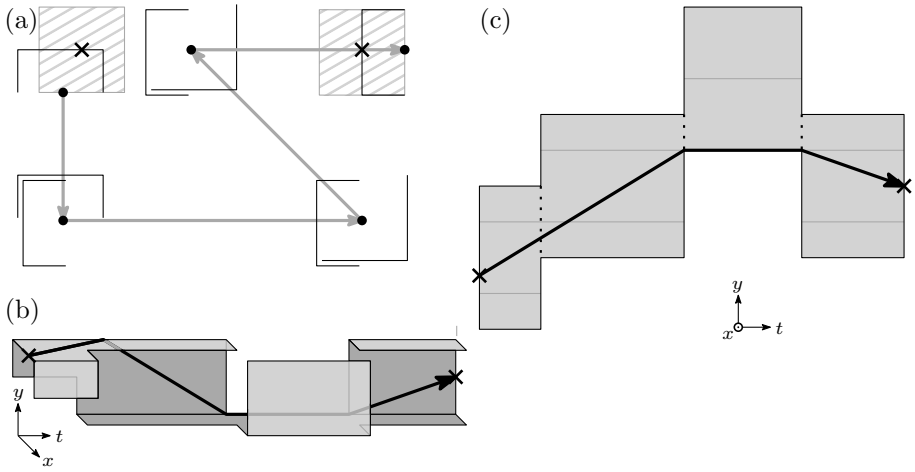


Figure 5.3: (a) A piecewise linear point trajectory (in gray, with dots as vertices) with given labels (in gray, hatched, centers marked by crosses) at the endpoints. (b) The corresponding configuration space of allowed label positions for $\alpha = \pi/2$. (c) An unfolding of (b) and a shortest path through it. Portals are shown as dotted line segments.

portal $\Psi(t'_2)$, passing through all intermediate portals in sequence. We will now argue that the shortest such path produces the most desirable label trajectory.

Lemma 5.1. *A shortest path from $\Psi(t'_1)$ to $\Psi(t'_2)$ through R yields a label trajectory minimizing both (i) the average speed and (ii) the maximum speed of p's label relative to p.*

Proof. (i) Let π be such a shortest path, which must be a t-monotone polygonal chain. By $T(\pi)$ we will denote the summed length of the projections onto the t-axis of the links of π, and by $Y(\pi)$ the same quantity for the y-axis. Every t-monotone path π' from $\Psi(t'_1)$ to $\Psi(t'_2)$ must traverse the same distance in the t-direction, namely $T(\pi') = T(\pi) = t'_2 - t'_1$. Since π is the shortest such path, it therefore minimizes the distance traveled in y-direction, that is, $Y(\pi) \leqslant Y(\pi')$ for all π'. Thus π minimizes the average speed $Y(\pi)/T(\pi)$ of p's label.

(ii) Let ab be a steepest link of π, that is, ab has the maximum absolute slope among the links of π. As in Figure 5.4(b), suppose that ab's projection onto the t-axis is $[t', t'']$ and that ab has negative slope (the case where it has positive slope is similar). Then π cannot make a left turn at a, as that would make ab less steep than the link preceding it. So π must either make a right turn at a, or start at a. If π makes a right turn at a, then a must be the bottom endpoint of portal $\Psi(t')$, as we could otherwise shorten π by a downward deformation at a, as in Figure 5.4(a). Since a lies above b, this makes ab the shortest path not

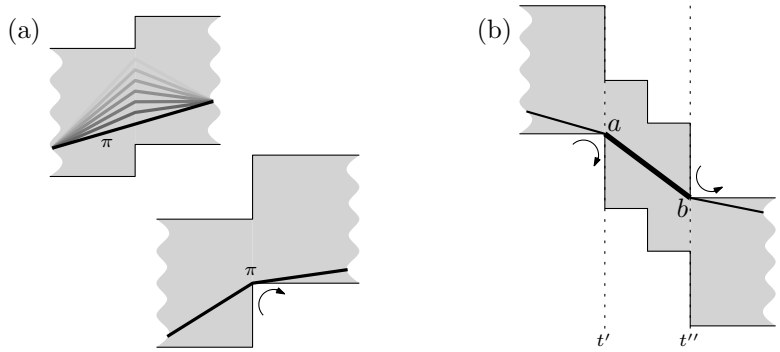


Figure 5.4: (a) *Top:* Any (gray) path through R containing a right turn somewhere other than the bottom of a portal can be shortened (into the black path). *Bottom:* Thus, a shortest path π must have all its right turns at the bottoms of portals (and all its left turns at the tops of portals). (b) Therefore, the steepest link ab of π is the least steep connection between its two portals.

only from a to b, but also from $\Psi(t')$ to b. If π instead starts at a, then $t'_1 = t'$, so the same condition holds. Reasoning symmetrically for b yields that ab must also be the shortest path from a to $\Psi(t'')$. We conclude that ab is the shortest path from $\Psi(t')$ to $\Psi(t'')$. This implies that any t-monotone path from $\Psi(t')$ to $\Psi(t'')$ must be at least as steep as ab somewhere between t' and t''. Thus π minimizes the maximum speed $Y(ab)/T(ab)$ of p's label. □

Theorem 5.2. *Suppose we are given a dynamic point set P with n points, along with static labelings $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ of it at times t_1 and t_2. For each $p \in P$ let k_p be the number of vertices in its polygonal trajectory between times t_1 and t_2. In $O(\sum_{p\in P} k_p)$ time and $O(\max\{k_p \mid p \in P\})$ space we can then compute a dynamic labeling $\mathcal{L}(t_1, t_2)$ respecting $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$, that for each point minimizes both its average and its maximum label speed.*

Proof. Lemma 5.1 shows that $\mathcal{L}(t_1, t_2)$ can be obtained by computing a shortest path through the unfolded configuration space for each point. It remains to show that this can be done in $O(k_p)$ time and space for a point $p \in P$ with $\mathit{life}(p) \cap [t_1, t_2] = [t'_1, t'_2] \neq \emptyset$. As before, let R denote the simple, rectilinear polygon with $O(k_p)$ vertices that is p's unfolded configuration space. Through R we seek either a shortest point-to-point path (if $\mathit{life}(p) \supseteq [t_1, t_2]$), a shortest edge-to-edge path (if $\mathit{life}(p) \subset [t_1, t_2]$), or a shortest point-to-edge path (otherwise). This can be done in linear time by Theorem 2.4. □

5.2.2 Hourglass trimming

In the previous section we described an algorithm to minimize both the average and the maximum speed of all labels in a dynamic labeling that interpolates between two given static labelings. As the experimental evaluation in the next section will show, however, high label speeds can still occur when a poor choice of static labelings is made. To see why this occurs, consider the example with $\alpha = \pi/2$ in Figure 5.5. Recall that we compute static labelings at regular intervals of Δt time. In the example, point p's trajectory makes a sharp left turn at time $t+\varepsilon$ for some small $\varepsilon > 0$. The static labeling algorithm completely disregards this when computing the labeling for time t, and comes up with the depicted leftmost label position for p. Now whatever labeling is picked for time $t + \Delta t$, the label for p will have to move substantially within ε time units. The problem here is that the chosen label position is just in the range of positions considered to be behind the point at time t, but the same position lies quite a bit in front of the point at time $t + \varepsilon$.

To fix this, we shall restrict the range of label positions that the static labeling algorithm is allowed to use, based on the trajectories of the points. We will need some definitions first. Consider a fixed point p, and let R be its unfolded configuration space. Let $\pi(a, b)$ denote the shortest path in R from a to b, for any $a, b \in R$. For a time interval $[t', t'']$ in which p is alive, we define the *hourglass* $H(t', t'')$ as the region enclosed by $\pi(a, c)$ and $\pi(b, d)$, where the segments ab and cd are the intersections of R with the vertical lines at t' and t''. Of the two paths $\pi(a, c)$ and $\pi(b, d)$ we call the upper one the hourglass's *upper chain*, and the other its *lower chain*. If the two chains intersect, then their intersection is a common subchain called the *string*. The hourglass is then called *closed*, and removal of the string leaves two connected components called

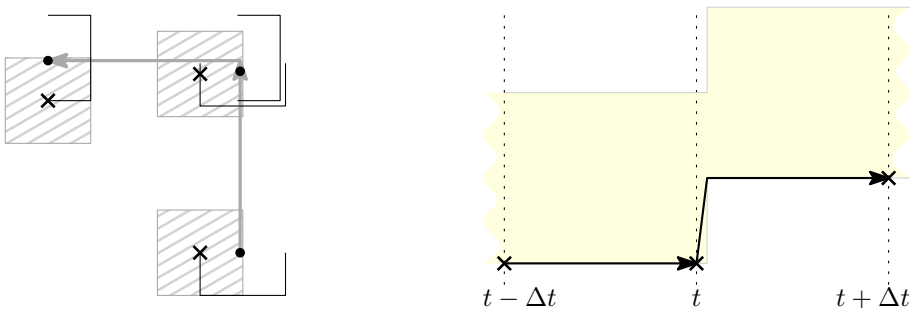


Figure 5.5: An example of the static labeling algorithm making a poor choice with regards to the dynamic labeling. On the left a point receives a leftmost label just before making a sharp left turn, on the right the configuration space shows that even the slowest label trajectory must then contain a quick motion.

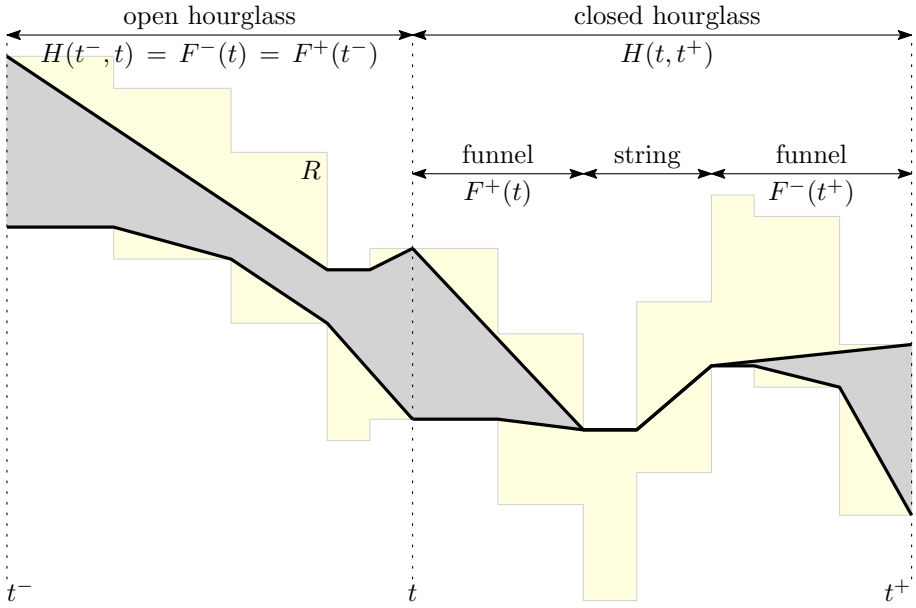


Figure 5.6: An example of an open hourglass (left) and a closed hourglass (right), the latter consisting of two funnels connected by a string.

funnels. If the hourglass is not closed, it is called *open*. Figure 5.6 illustrates these concepts

Now consider a time instance $t \in \mathit{life}(p)$. Let $t^- = \max\big(\mathit{birth}(p), t - \Delta t\big)$, let $t^+ = \min\big(t + \Delta t, \mathit{death}(p)\big)$, and consider the hourglasses $H(t^-, t)$ and $H(t, t^+)$. Whatever label positions are chosen at times t^-, t, and t^+, the slowest interpolation between them (that is, the shortest path connecting them in R) must stay within the union of these two hourglasses. If $H(t^-, t)$ or $H(t, t^+)$ has steep edges, as in the example of Figure 5.6, then a fast moving label may result. We shall therefore narrow the ranges of valid label positions in such a way that these steep edges are "trimmed off" the hourglasses. If $H(t^-, t)$ is closed, then let $F^-(t)$ denote the rightmost funnel of $H(t^-, t)$, and let $F^-(t) = H(t^-, t)$ otherwise. Similarly, let $F^+(t)$ denote either $H(t, t^+)$ (if open) or its leftmost funnel (if closed). We assume we are given a parameter v that denotes a speed deemed reasonable for labels. We now translate a line with slope $+v$ down from positive infinity along the y-axis until it has become tangent to one of the two chains defining $F^-(t)$. Similarly, we translate a line with slope $-v$ up from negative infinity until it has become tangent to one of the two chains defining $F^-(t)$. These two lines define a narrower interval on the vertical line at t—see Figure 5.7(a). If we apply the same procedure at time t^- (using $F^+(t^-)$), then the new hourglass $H'(t^-, t)$ between the two narrowed intervals at t^- and t is

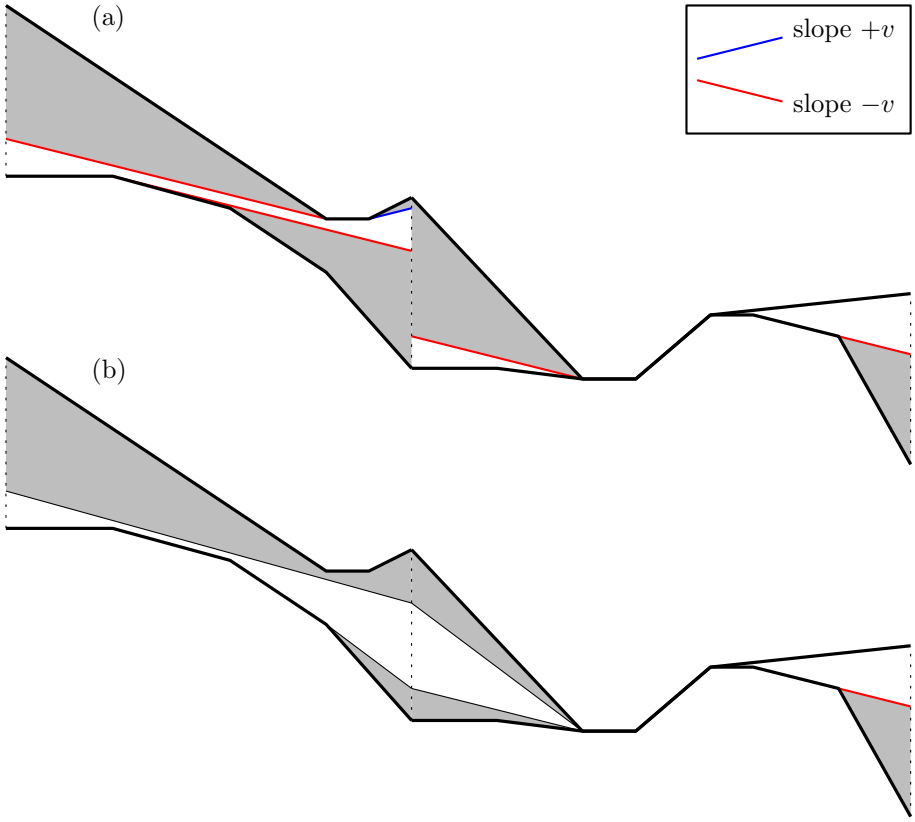


Figure 5.7: An example of hourglass trimming on the hourglasses of Figure 5.6. (a) The trimmed hourglasses in isolation, shown in white. (b) When necessary, the trimmed hourglasses are modified so that they connect to each other.

the trimmed hourglass we are after. Specifically, the slowest label interpolation through $H'(t^-, t)$ cannot exceed the speed v, except in two cases:

- $H'(t^-, t)$ is closed, and its string contains an edge steeper than v, as on the right in Figure 5.7(a). In this case there is nothing that can be done to avoid exceeding the speed v with p's label.
- $H'(t^-, t)$ is open, and a bi-tangent of its upper and lower chain is steeper than v, as on the left in Figure 5.7(a). In this case it might be possible to trim the hourglass further, but sometimes this will simply result in a closed hourglass, making the previous case apply. Hence, we decided not to trim the hourglass further. Our experiments described in the next section show that our current method works quite well in practice.

In the same way, we compute the trimmed hourglass $H'(t, t^+)$, using $F^+(t)$ and $F^-(t^+)$. Typically, the narrowed interval at t that defines $H'(t^-, t)$ will differ from the narrowed interval at t that defines $H'(t, t^+)$. If they overlap this poses no problem, as we may then simply take their intersection. In Figure 5.7(a), however, they are disjoint. In that case, there is nothing we can do to avoid high label speeds on both sides of t. We then take the interval in between the two, as in Figure 5.7(b). Note that this can undo some of the trimming, making the hourglass wider again.

5.3 Experimental evaluation

We will now evaluate the effect of varying the timestep parameter Δt on the quality of the produced labeling, that is, on the number of free labels and on the speeds at which labels move. Intuitively, one would expect both the number of free labels and the label speeds to increase as the timestep approaches 0. Thus there is a trade-off between how many labels are free and how slow the labels move. Our main goal is to quantify this trade-off experimentally.

5.3.1 Computation time

We have measured the computation time of our C++ implementation only informally, just to ensure it was fast enough for use in interactive applications. On the modest hardware used for our experiments (an Intel Q6600 2.40GHz with 3GB RAM running Ubuntu 12.10), Interpolation takes about 0.4 milliseconds to interpolate between two 100-point labelings, and 2.2 milliseconds between two 1,000-point labelings. Producing such labelings with FourGreedySweeps takes about 5 milliseconds for 100 points, and 189 milliseconds for 1,000 points. Note that this is with a simple $O(n^2)$-time implementation, and not the more sophisticated $O(n \log n)$-time implementation described in Section 4.1.3. The latter could no doubt label 1,000 points much more quickly, but such large numbers of labels cannot be readably displayed on a reasonably sized screen anyway. The extra engineering effort was therefore deemed unnecessary.

5.3.2 Experimental setup

To evaluate the quality of the produced dynamic labelings, we have used a network of streets in the Dutch city of Eindhoven (see Figure 5.8). Moving points were created to move along five polygonal paths through the network, at constant and equal speeds of 35 px/s. The arrival times of the points on each route were created by a Poisson process with parameter 5 s. This makes the time between arrivals of successive points on a route an exponentially distributed random variable with a mean of 5 s. Different seeds for the random number

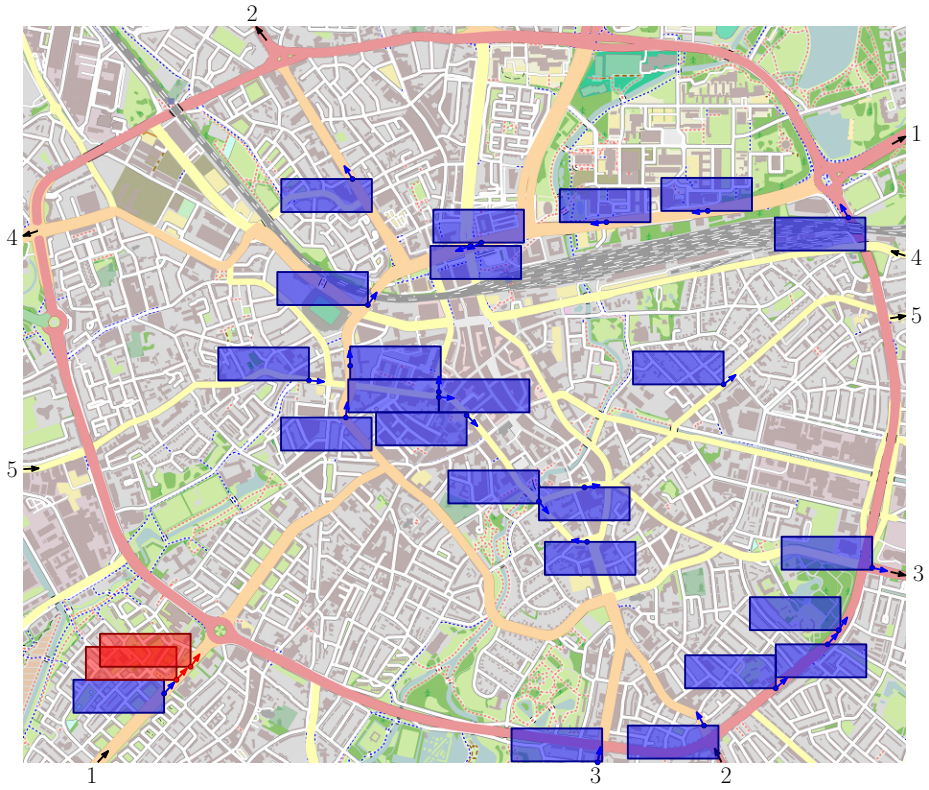


Figure 5.8: The road network used for our experiments, and some labeled points moving across it along five routes. Blue labels are free, red labels are not.

generator thus create different problem instances. We created 100 of these, and labeled all of them in the α-trailing model with $\alpha = \pi/2$. For polygonal point trajectories this is the only value of α that makes sense. With $\alpha > \pi/2$ the range of allowed label positions is so narrow that the ranges before and after a sharp turn may no longer overlap, making a continuous labeling impossible. With $\alpha < \pi/2$ the ranges are so wide that they may overlap in two places during a sharp turn, making it necessary to use a more complicated $O(n \log n)$-time algorithm to find shortest paths through the configuration space. For curved trajectories in which a point's direction changes continuously these problems disappear, making any value of α in the range $[0, \pi]$ viable. We decided to use polygonal trajectories mainly for simplicity of the implementation. In principle our algorithms can also be applied to curved trajectories, but this makes the unfolded configuration space a *splinegon* instead of a rectilinear polygon, requiring

a more intricate implementation of the shortest path algorithm (as discussed in Section 2.2.1).

For each of the 100 problem instances we used our algorithm to produce dynamic labelings from $t = 0$ to $t = t_{\max} = 60$ s for several different values of the timestep Δt. To determine the quality of such a dynamic labeling $\mathcal{L}$ we did not compute the exact intervals of time during which each label was free. Instead, $\mathcal{L}$ was sampled at regular times at a rate of 25.6 samples per second (1537 samples total over 60 seconds). This is roughly the same framerate as used in movies (24 frames per second), but with the time between samples changed to have a finite binary floating point representation (from $1/24$ s $= 5/120$ s to $5/128$ s). For each sample we recorded the number of free and non-free labels, as well as the amount each label moved (relative to its point) since the last sample. In addition, we recorded the *free label area* for each sample: the area covered by the union of the labels, minus the area covered by more than one label. All of this was done for thirteen different timesteps, the lowest being $\Delta t = 1/25.6$ s ≈ 0, so that each sample is labeled independently without regard for label speed, and the highest being $\Delta t = 61$ s > 60 s, so that label speeds are minimized without regard for label freeness. The graphs below offer a summary of the resulting data. The software used to generate the data and the graphs can be downloaded from `http://dirkgerrits.com/programming/flying-labels/`.

5.3.3 Results at a glance

The graphs in Figures 5.9 and 5.10 provide a high-level overview of the quality of labelings computed by our algorithm. Both figures show three graphs. The top graph shows how the label speed (measured in pixels per second), averaged over all moving points and over the whole time interval $[0, t_{\max}]$, decreases as we pick higher and higher values for Δt. The middle graph shows the same for the fraction of the labels that are completely free. The bottom graph, lastly, shows the free label area divided by the total area that would be spanned by the labels if they did not overlap.

Figure 5.9 displays only the minimum and maximum (dotted lines), 25% and 75% quantiles (dashed lines), and mean (solid line) over the 100 problem instances. The red lines show the results of the algorithm without hourglass trimming, the black lines show the results when hourglass trimming is used with a parameter of $v = 10$ px/s.

In both cases we see a very sharp decline in label speeds until around $\Delta t = 2$ s, with a more modest corresponding decrease in label freeness. For higher Δt the decrease in label speeds slows down substantially while the decrease in label freeness continues. Thus, these preliminary results suggest that a timestep of around 2 seconds should yield good labelings.

The effect of turning hourglass trimming on is similar to that of increasing the timestep: the label speeds and freenesses both decrease. In this sense it

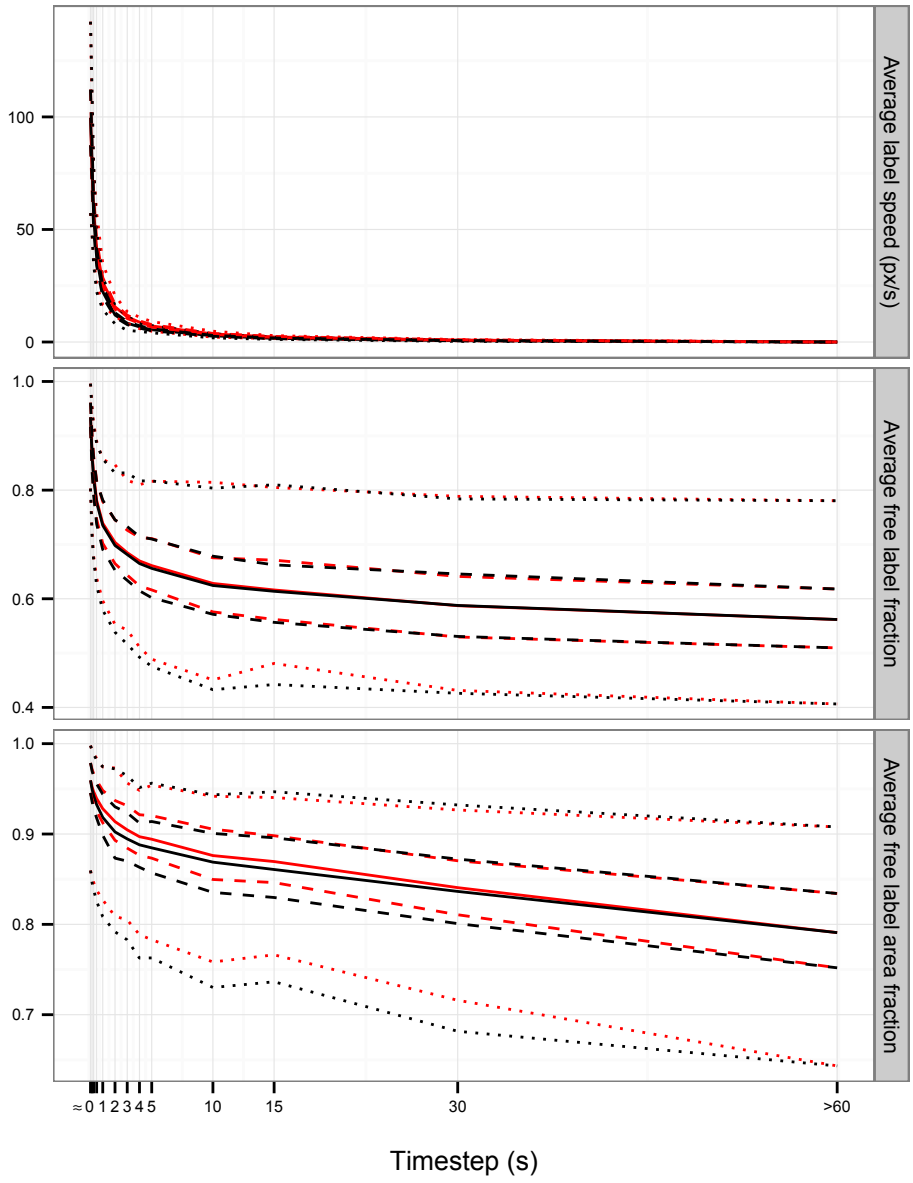


Figure 5.9: The effects of varying the timestep from $\Delta t \approx 0$ to $\Delta t > 60$ s on label speeds, number of free labels, and free label area, both with (black) and without (red) hourglass trimming. Shown are the minimum and maximum (dotted lines), 25% and 75% quantiles (dashed lines), and mean (solid line) over the 100 problem instances. Figure 5.10 shows the problem instances individually.

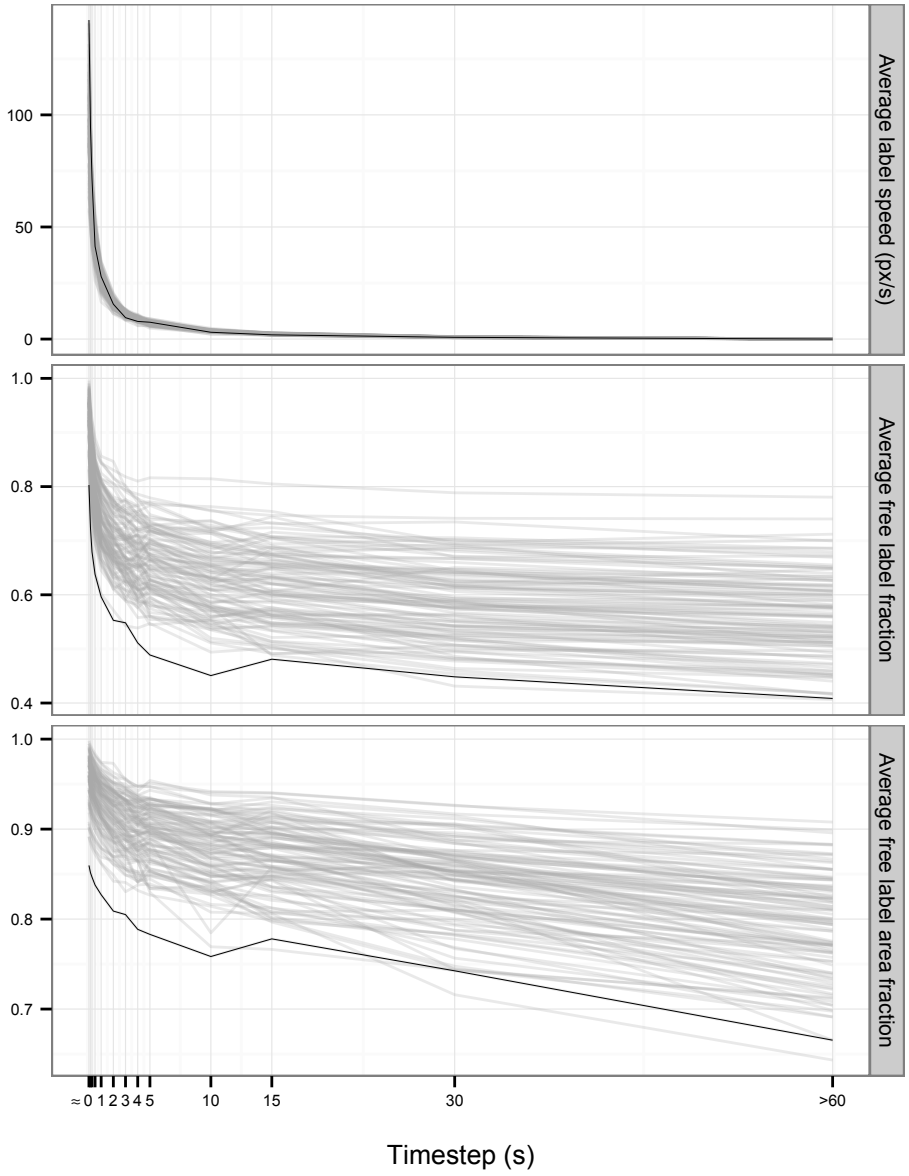


Figure 5.10: The effects of varying the timestep from $\Delta t \approx 0$ to $\Delta t > 60$ s on label speeds, number of free labels, and free label area, when hourglass trimming is not used. (When hourglass trimming is used, the graphs look very similar but shifted slightly downward.) All 100 problem instances are shown, with one particularly "difficult" instance highlighted in black. This problem instance is explored more in the following figures.

forms an alternative to raising the timestep. Hourglass trimming does something more, however, as we shall see next when we examine a single problem instance in more detail. In particular, we will look at the problem instance highlighted in black in Figure 5.10. This figure shows the same graphs as Figure 5.9, but instead of summary statistics it shows each of the 100 problem instances as its own polyline. The highlighted instance seems fairly hard to label well with our algorithm, as indicated by its poor label freeness scores.

5.3.4 Detailed analysis of a difficult instance

Figures 5.11 through 5.14 provide a closer look at the "difficult" problem instance highlighted in Figure 5.10. In Figure 5.11 thirteen graphs show how the label speeds of the individual moving points develop over time for the thirteen different values of Δt that were used. Each of the moving points is drawn as a polyline, showing its speed at each time sample. The red lines show the situation when hourglass trimming is not used. In the case of $\Delta t \approx 0$ almost all labels move rather violently (as was to be expected). For the other values of Δt, and especially the higher ones, a curious pattern appears. Label movement tends to be slow overall and decrease when Δt increases, but there are "spikes" of very high label movement near times that are a multiple of Δt. This effect is caused by a point changing direction near a multiple of Δt, as was explained in Section 5.2.2. This was our motivation for introducing hourglass trimming. The black lines show what happens when hourglass trimming is employed with the parameter $v = 10$ px/s. The regularly occurring spikes vanish, leaving label speeds with less variance. Note that the resulting label speeds do still exceed v, and by quite a bit for smaller timesteps. Figure 5.12 shows the label speeds of this problem instance in a different way. Here the y-axis shows how common the speeds on the x-axis are, so slower speeds show up as the "mass" under the curve shifting to the left.

Figures 5.13 and 5.14 show how the free label count and area vary during this particular problem instance. Here we also see peaks near times that are a multiple of Δt, although they are less pronounced. These correspond to large numbers of free labels at the times where static labelings are computed, which then quickly become non-free due to the interpolation. These peaks are also lessened by hourglass trimming. This is not necessarily a good or bad effect. When labels are free only for a split second one cannot read them anyway, so such peaks do not really "count". Of course it is unfortunate that such peaks were not plateaus to begin with, having more consistently free labels, but that is not a problem of hourglass trimming. Hourglass trimming does lower some of the plateaus that do occur, making it a trade-off instead of a clear improvement. Given its drastic reduction of high label speeds, however, it seems a particularly worthwhile trade-off.

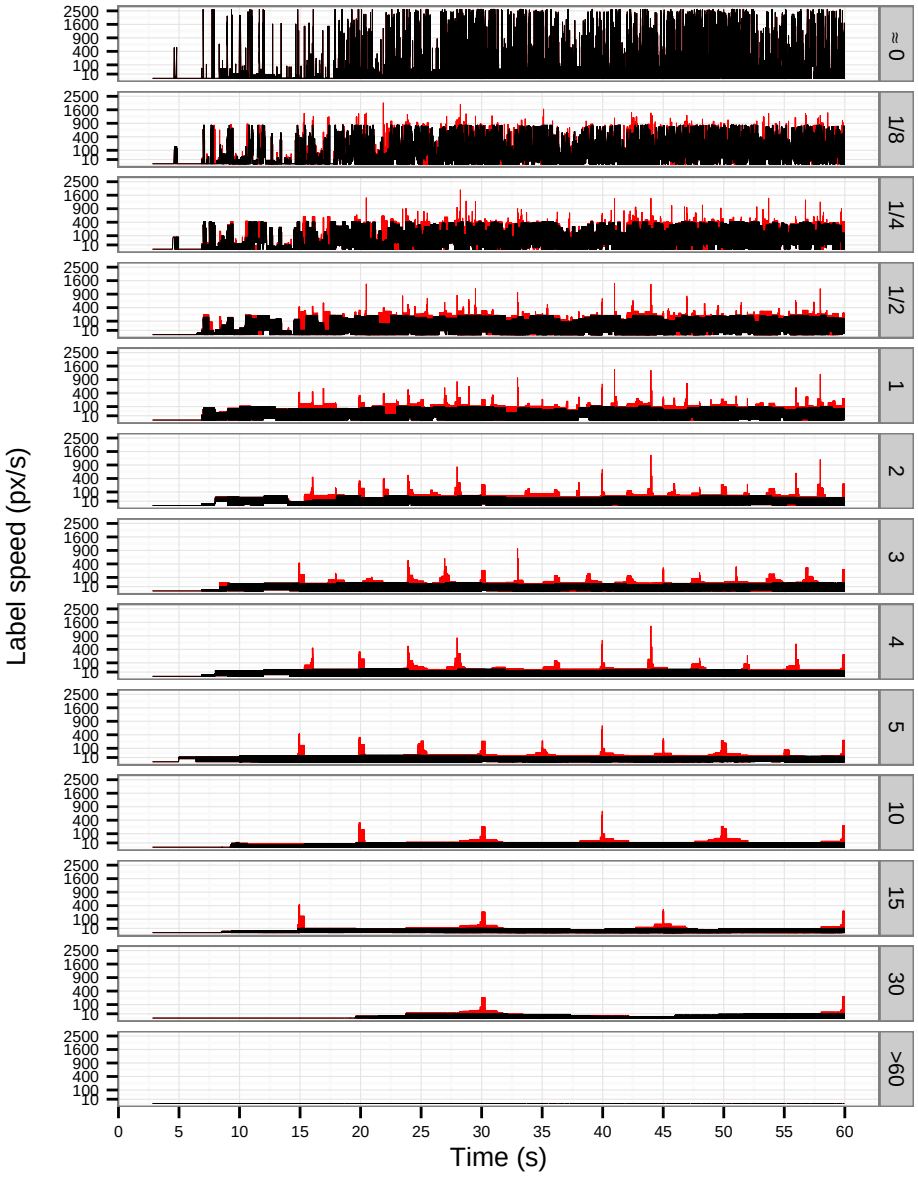


Figure 5.11: The progression of label speeds over one particular problem instance, for thirteen different timesteps, both with (black) and without (red) hourglass trimming. Note that the y-axis uses a square root scale. With a linear scale the higher timesteps would have indistinguishably low speeds, while with a logarithmic scale the lower timesteps would have indistinguishably high speeds.

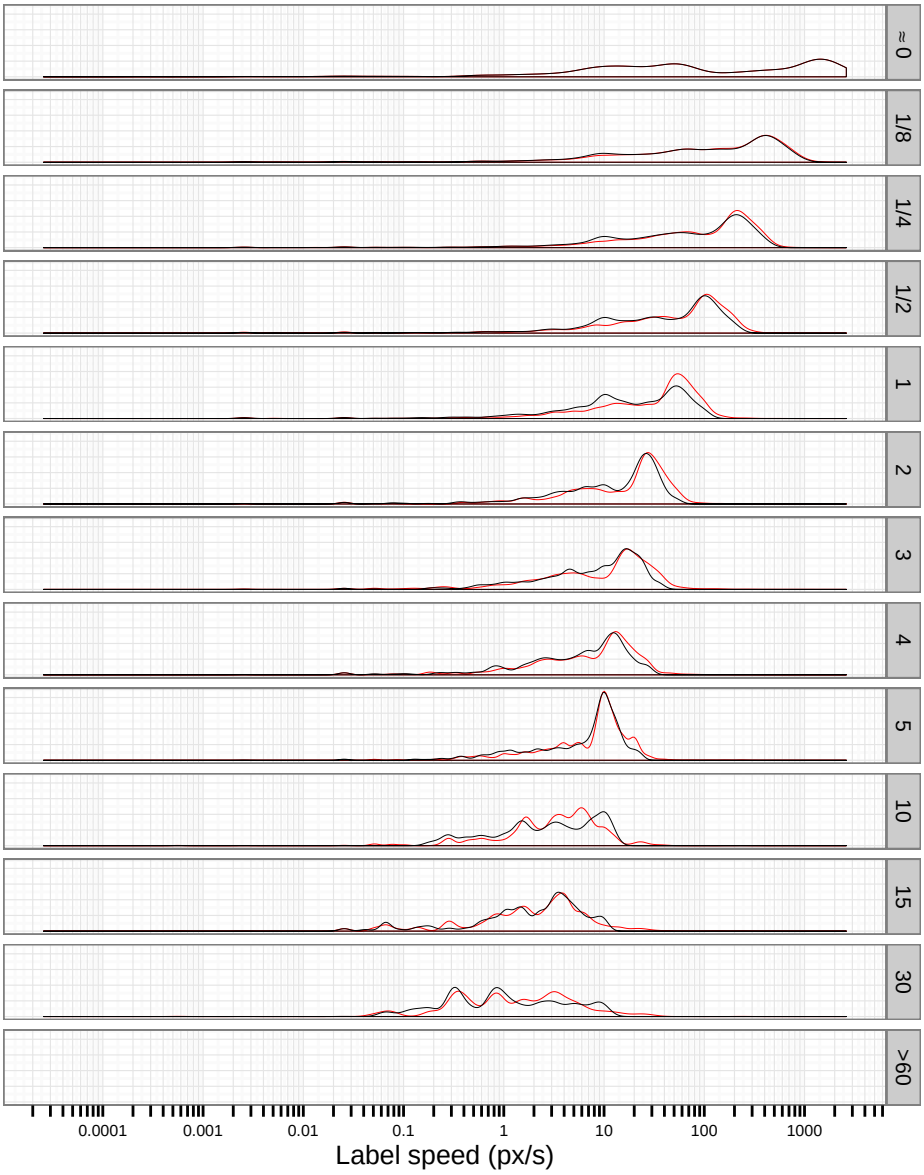


Figure 5.12: Density of the number of occurrences of each label speed in one particular problem instance, for thirteen different timesteps, both with (black) and without (red) hourglass trimming. Note that the x-axis uses a logarithmic scale.

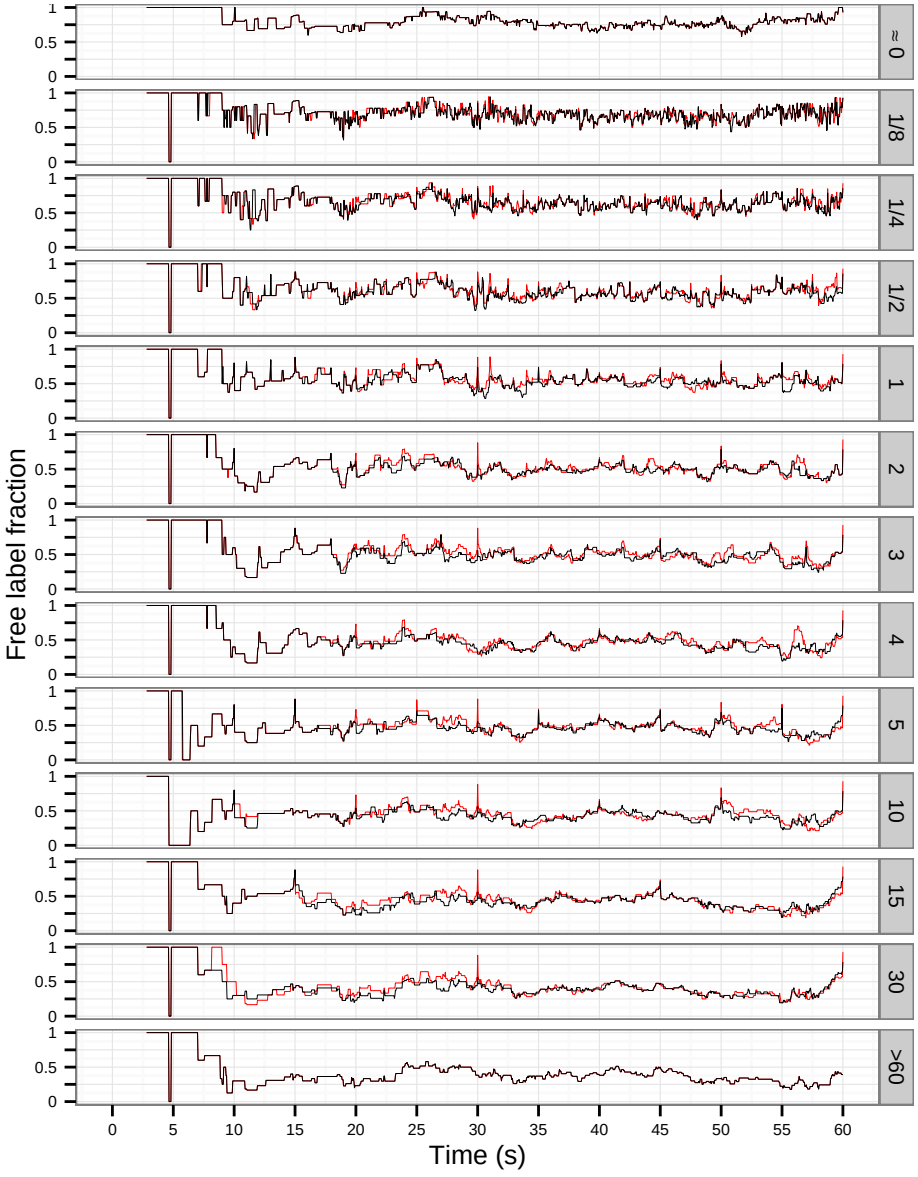


Figure 5.13: The progression of the ratio of free labels over one particular problem instance, for thirteen different timesteps, both with (black) and without (red) hourglass trimming.

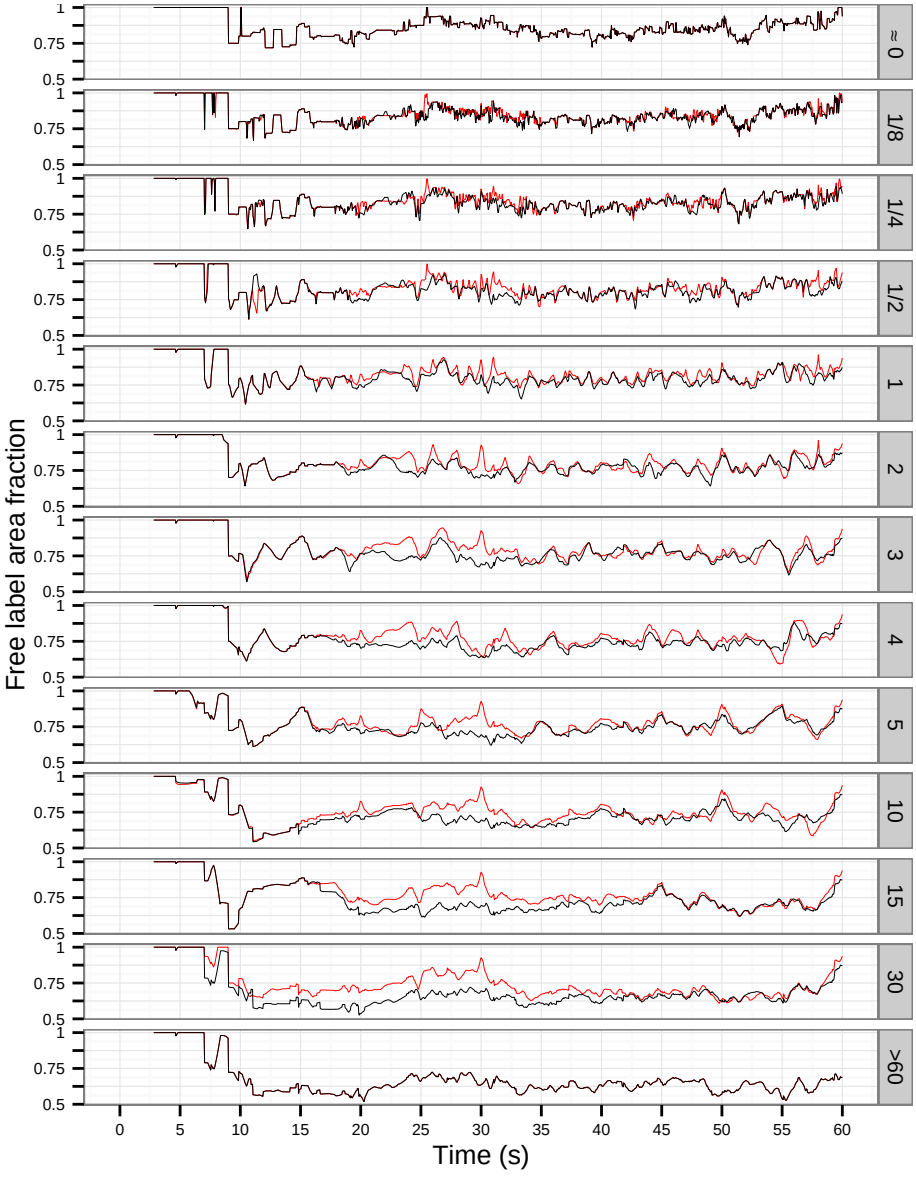


Figure 5.14: The progression of the ratio of free label area over one particular problem instance, for thirteen different timesteps, both with (black) and without (red) hourglass trimming.

5.4 Discussion and future research

In this chapter we developed a heuristic algorithm for free-label maximization on dynamic point sets, and evaluated it experimentally. The algorithm has been presented with the assumption that all points move on polygonal trajectories, but could be implemented just as well for curved trajectories. Instead of operating on polygons our algorithms will then work with curved *splinegons*. This change can be effected using techniques from the literature, as was discussed in Section 2.2.1. We presented and evaluated our algorithm in a label model that we call the α-trailing model, a subset of the 4-slider model in which the label needs to stay behind its moving point.

Our algorithm works by computing static labelings with many free labels at regular intervals using an algorithm of the previous chapter, and then interpolating between these static labelings in a way that minimizes both the average and the maximum label speed. By varying the time between static labelings one obtains a trade-off between the number of free labels over time, and the speeds of the labels. The trade-off seemed favorable in experiments, in the sense that a small increase in label speeds can yield a great improvement in the number of free labels.

With these preliminary results we have only scratched the surface. From a theoretical point of view, algorithms with proven approximation ratios are still sorely lacking for dynamic labeling. Our initial aim was to develop such an algorithm for the average number of free labels over time. So far this goal has remained illusive. From a practical point of view, there is room for improvement in other directions. While hourglass trimming was a step in the right direction, our method for choosing static labelings can undoubtedly be improved further. Ideally, the high number of free labels it achieves should not drop off so quickly when interpolation is applied. Additionally, there are many more experiments we could perform. We could look at more varied problem instances, and explore how the quality of the computed labelings depends on the density of the point set to be labeled. We may also measure additional metrics, such as counting labels as free only when they remain so for more than a second. Since label motion is more noticeable for isolated labels and near the center of view than it is for overlapping labels and at the periphery [73], we may try to measure these separately and try to adjust our algorithm to exploit this effect. Ultimately, though, to determine how well our algorithm (and its possible enhancements) *really* perform we will have to do user studies to go along with these measurements.

Conclusion 6

In this thesis we looked at two classes of problems. The first concerns pushing an object to a desired location with a robot, the second concerns a set of moving points, each pulling a textual label along with it. When objects are too heavy or too large to carry, pushing them can be the only way to move them. The challenge is then to compute a trajectory for the robot that causes it to push the object correctly. Labeling sets of moving points is especially interesting in applications such as air-traffic control and car navigation systems. The challenge there is to compute trajectories for the labels so that they overlap each other as little as possible, while moving slowly and smoothly so that they can be read easily.

6.1 Computing push plans

Chapter 2 looks at the pushing problem. We re-examine one of the state-of-the-art algorithms for this problem, which is due to Nieuwenhuisen et al. [69]. Their algorithm handles the case of a disk-shaped pusher moving a disk-shaped object among a set of n non-intersecting line segment obstacles in the plane. At a high level their algorithm is based on rapidly-exploring random trees (RRT) [54], and builds up a tree of feasible pushing motions with the initial object position at the root and eventually connecting to its destination position in a leaf. To build the edges of this tree they use a subroutine that computes a push plan for a given object path consisting of k line segments and circular arcs (rather than just given the object's destination). Their implementation of this subroutine has several shortcomings, however, which we were able to improve upon by using a different method based on constructing the configuration space of the problem. Firstly, we were able to do away with the need to preprocess the obstacles in $O(n^2 \log n)$ time into an $O(n^2)$-space data structure. Secondly, our algorithm can handle more general curved paths, and can easily be extended to handle more general obstacles as well (convex pseudodisks, namely). Thirdly, if there

exists a contact-preserving push plan, in which the pusher maintains contact with the object at all times, then we can compute the shortest such push plan. We have shown, however, that for some obstacle configurations the pusher will have to let go of the object on occasion to resume pushing from a different location. We can also compute such *unrestricted push plans*, although we can then not guarantee finding the shortest one. Computing shortest unrestricted push plans is an obvious direction for future research.

Of more practical interest, perhaps, is handling more complicated shapes of the object and pusher. The way the object then responds to being pushed is much more complex and hard to predict (see Section 1.1.1). Assuming we do know this, however, the main added challenge is that the orientations of the object and pusher now matter. Lynch and Mason [59] discuss conditions under which the relative orientation of the pusher and object remain fixed during contact-preserving pushing, simplifying the problem somewhat. Unfortunately, our method based on a 2-dimensional configuration space would still not work. If we are content with contact-preserving push plans we would still have to add a dimension for the relative orientation of the pusher and the object, and if we desire unrestricted push plans we would need extra dimensions for both their orientations.

Taking a step back out of the context of the above RRT-algorithm, we might also investigate alternative approaches to the main problem. Instead of using a subroutine to find push plans for a given object path, perhaps the tasks of pushing to a given destination can be tackled more directly. If this is done with the configuration-space approach, then one would again gain an extra dimension. Moreover, not just any path through this space would result in a valid push plan, complicating matters further.

6.2 Dynamic point labeling

Chapters 3 through 5 look at labeling of dynamic point sets, in which points are added and removed and in which points move on continuous trajectories. In Chapter 3 we examine the complexity of dynamic point labeling. Most instances of static point labeling are NP-hard, making their generalizations to dynamic point sets NP-hard as well. Intuitively, however, dynamic point labeling should be even harder still. We prove that deciding whether all points in a dynamic point set can be labeled without overlap is strongly PSPACE-complete, even with unit-square labels. This immediately implies that all natural optimization problems for dynamic point labeling are strongly PSPACE-hard. We were able to show this even when the dynamic nature of the point set is greatly restricted. For example, only one point may be moving, or in a set of unmoving points a single new point is added while a single old point is removed. Similarly, instead of the motion and (dis)appearance of the points being arbitrary, it may be the result of panning, rotating, or zooming a finite viewport onto an interactive map

and our results will still apply. For the specific optimization problem of labeling all points without overlap with labels of maximum size we show that one cannot even obtain a polynomial-time approximation scheme unless P = PSPACE.

In Chapters 4 and 5 we introduce and study the free-label maximization problem, a new variant of the labeling problem. For cartography one is interested in number maximization, where overlap of labels is prohibited and the goal is to assign labels to a maximum number of the points. In free-label maximization one instead labels all the points, placing them so as to maximize the number of free labels (labels which are not overlapped by other labels). While less useful for the creation of static maps, free-label maximization is a natural fit for the labeling of moving points. Generalizing number maximization to moving points will cause labels to appear and disappear, which can be disturbing to the user. This is typically alleviated somewhat by fading them in and out, but free-label maximization avoids this problem entirely. User studies have found that for some tasks momentary label overlap is better than sudden changes in the labeling to avoid it [3], and that label motion is less noticeable for overlapping labels [73]. In air-traffic control, free-label maximization is even mandated, more or less, as European safety regulations require that all airplanes are labeled at all times [25].

As a first step towards free-label maximization on dynamic point sets, Chapter 4 studies the problem for a set of n static points. For unit-square labels placed to touch the points, we develop constant-factor approximation algorithms that run in $O(n \log n)$ time, as well as polynomial-time approximation schemes. In Chapter 5 we use such a $O(n \log n)$-time static labeling algorithm as a subroutine in a dynamic labeling algorithm. For labels that trail behind their moving points we propose a heuristic algorithm that computes static labelings at regular intervals, and then interpolates between them in a way that minimizes both the average and the maximum label speed. By varying the time between static labelings we hope to obtain a trade-off between the number of free labels over time and the speeds of the labels. This trade-off has been confirmed in experiments, and a small increase in label speeds was found to yield a great improvement in the number of free labels.

Directions for future research abound for dynamic point labeling. No published algorithms so far achieve a proven approximation ratio for dynamic point labeling in its most general case. Changing this state of affairs would be a very nice accomplishment from a theoretical point of view. From a practical point of view, better and more sophisticated heuristics are needed. User studies indicate that label motion is much more noticeable for isolated labels and near the center of the display than it is for overlapping labels and at the periphery of the display [73]. Presumably one could use this effect to keep labels free for longer periods, without a perceived increase in label speeds.

References

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:209–218, 1998.

[2] P. K. Agarwal, J. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proceedings of the 1997 IEEE International Conference on Robotics & Automation (ICRA'97)*, volume 4, pages 3124–3129. 1997.

[3] K. R. Allendoerfer, J. Galushka, and R. H. Mogford. Display system replacement baseline research report. Technical report DOT/FAA/CT-TN00/31, William J. Hughes Technical Center, Atlantic City International Airport (NJ), 2000.

[4] H. Arai and O. Khatib. Experiments with dynamic skills. In *Proceedings of the 1994 Japan-USA Symposium on Flexible Automation*, pages 81–84. 1994.

[5] I. J. Balaban. An optimal algorithm for finding segment intersections. In *Proceedings of the 11th Annual ACM Symposium on Computational Geometry (SoCG'95)*, pages 211–219. 1995.

[6] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 747–756. 1997.

[7] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, 2006.

[8] K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry: Theory and Applications*, 43(3):312–328, 2010.

[9] B. Bell, S. Feiner, and T. Höllerer. View management for virtual and augmented reality. In *Proceedings of the 14th Annual ACM Symposium on*

User Interface Software and Technology (UIST'01), pages 101–110. ACM, 2001.

[10] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[11] M. de Berg and D. H. P. Gerrits. Approximation algorithms for free-label maximization. *Computational Geometry: Theory and Applications*, 45(4):153–168, 2012.

[12] M. de Berg and D. H. P. Gerrits. Computing push plans for disk-shaped robots. *International Journal of Computational Geometry & Applications*, 23(1), 2013. To appear.

[13] M. de Berg and D. H. P. Gerrits. Labeling moving points with a trade-off between label speed and label overlap. In *Proceedings of the 21st Annual European Symposium on Algorithms (ESA'13)*. 2013. To appear.

[14] P. Berman, B. DasGupta, S. Muthukrishnan, and S. Ramaswami. Efficient approximation algorithms for tiling and packing problems with rectangles. *Journal of Algorithms*, 41(2):443–470, 2001.

[15] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Computational Geometry: Theory and Applications*, 9(3):159–180, 1998.

[16] K. Buchin and D. H. P. Gerrits. Dynamic point labeling is strongly PSPACE-hard. In *Proceedings of the 29th European Workshop on Computational Geometry (EuroCG'13)*, pages 241–244. 2013.

[17] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, pages 892–901. 2009.

[18] T. M. Chan. A note on maximum independent set in rectangle intersection graphs. *Information Processing Letters*, 89:19–23, 2004.

[19] T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48:373–392, 2012.

[20] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

[21] H. Cords, M. Luboschik, and H. Schumann. Floating labels: improving dynamics of interactive labeling approaches. In *Proceedings of the IADIS International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing (CGVCVIP'09)*, pages 235–238. 2009.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[23] S. van Dijk, D. Thierens, and M. de Berg. Using genetic algorithms for solving hard problems in GIS. *GeoInformatica*, 6(4):381–413, 2002.

[24] Y. Djouadi. Cartage: A cartographic layout system based on genetic algorithms. In *Proceedings of the 5th European Conference and Exhibition on Geographical Information Systems (EGIS'94)*, pages 48–56. 1994.

[25] A. Dorbes. Requirements for the implementation of automatic and manual label anti-overlap functions. EEC Note No. 21/00, EUROCONTROL Experimental Centre, 2000.

[26] A. Duverger. Development of a mathematical weighted formula to eliminate the overlapping of aircraft labels on the ATC radar display. EEC Note No. 19/05, EUROCONTROL Experimental Centre, 2005.

[27] D. Ebner, G. W. Klau, and R. Weiskircher. Force-based label number maximization. Technical report TR-186-1-03-02, Vienna University of Technology, 2003.

[28] D. Ebner, G. W. Klau, and R. Weiskircher. Label number maximization in the slider model. In J. Pach, editor, *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, volume 3383 of *Lecture Notes in Computer Science*, pages 144–154. Springer Berlin/Heidelberg, 2005.

[29] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341–363, 1986.

[30] T. Erlebach, T. Hagerup, K. Jansen, M. Minzlaff, and A. Wolff. Trimming of graphs, with application to point labeling. *Theory of Computing Systems*, 47(3):613–636, 2010.

[31] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the 7th Annual ACM Symposium on Computational Geometry (SoCG'91)*, pages 281–288. 1991.

[32] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12:133–137, 1981.

[33] J. García-López and P. A. Ramos. A unified approach to conic visibility. *Algorithmica*, 28(3):307–322, 2000.

[34] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In F. Dehne, J. Iacono, and J.-R. Sack, editors, *Proceedings of the 12th International Symposium on Algorithms and Data Structures (WADS'11)*, pages 451–462. 2011.

[35] A. Gemsa, M. Nöllenburg, and I. Rutter. Sliding labels for dynamic point labeling. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG'11)*, pages 205–210. 2011.

[36] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2–4):210–225, 1993.

[37] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

[38] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.

[39] S. A. Hirsch. An algorithm for automatic name placement around point data. *Cartography and Geographic Information Science*, 9(1), 1982.

[40] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

[41] W. Huang, S.-H. Hong, and P. Eades. Effects of crossing angles. In *Proceedings of the 1st IEEE Pacific Visualization Symposium (PacificVis'08)*, pages 41–46. 2008.

[42] T. Igarashi, Y. Kamiyama, and M. Inami. A dipole field for object delivery by pushing on a flat surface. In *Proceedings of the 2010 IEEE International Conference on Robotics & Automation (ICRA'10)*, pages 5114–5119. 2010.

[43] H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983.

[44] E. Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[45] C. Iturriaga. *Map Labeling Problems*. Ph.D. thesis, University of Waterloo, Canada, 1999.

[46] M. Jiang, S. Bereg, Z. Qin, and B. Zhu. New bounds on map labeling with circular labels. In *Proceedings of the 15th Annual International Symposium on Algorithms and Computation (ISAAC'04)*, pages 606–617. 2004.

[47] T. J. Kang and P. Muter. Reading dynamically displayed text. *Behaviour & Information Technology*, 8(1):33–42, 1989.

[48] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete & Computational Geometry*, 1:59–70, 1986.

[49] M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, and J. Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Proceedings of the 2011 IEEE International Conference on Robotics & Automation (ICRA'11)*, pages 5722–5729. 2011.

[50] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.

[51] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[52] M. Lau, J. Mitani, and T. Igarashi. Automatic learning of pushing strategy for delivery of irregular-shaped objects. In *Proceedings of the 2011 IEEE International Conference on Robotics & Automation (ICRA'11)*, pages 3733–3738. 2011.

[53] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[54] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. 2001.

[55] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[56] M. Luboschik, H. Schumann, and H. Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1237–1244, 2008.

[57] S. Luke. *Essentials of Metaheuristics*. Lulu, 2nd edition, 2013.

[58] K. M. Lynch. Estimating the friction parameters of pushed objects. In *Proceedings of the 6th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 1, pages 186–193. 1993.

[59] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 15(6):533–556, 1996.

[60] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard CS, 1991.

[61] M. T. Mason. *Mechanics of Robotic Manipulation*. Intelligent Robots & Autonomous Agents. MIT Press, 2001. ISBN-10: 0-262-13396-2 ISBN-13: 978-0-262-13396-8.

[62] M. T. Mason, A. D. Christiansen, and T. M. Mitchell. Experiments in robot learning. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 141–145. 1989.

[63] M. T. Mason and K. M. Lynch. Dynamic manipulation. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS'93)*, pages 152–159. 1993.

[64] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 49–58. 1991.

[65] N. Miller and M. Sharir. Efficient randomized algorithm for constructing the union of fat triangles and of pseudodiscs, 1991. Unpublished manuscript.[1]

[66] J. L. Morrison. Computer technology and cartographic change. In D. R. F. Taylor, editor, *The Computer in Contemporary Cartography*. Johns Hopkins University Press, 1980. ISBN 0-471-27699-5.

[67] K. Mote. Fast point-feature label placement for dynamic visualizations. *Information Visualization*, 6(4):249–260, 2007.

[68] D. Nieuwenhuisen. *Path Planning in Changeable Environments*. Ph.D. thesis, Universiteit Utrecht, The Netherlands, 2007.

[69] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars. Pushing a disk using compliance. *IEEE Transactions on Robotics*, 23(3):431–442, 2007.

[70] M. A. Peshkin and A. C. Sanderson. Minimization of energy in quasi-static manipulation. *IEEE Transactions on Robotics & Automation*, 5(1):53–60, 1989.

[71] S. D. Peterson, M. Axholt, M. Cooper, and S. R. Ellis. Evaluation of alternative label placement techniques in dynamic virtual environments. In A. Butz, B. Fisher, M. Christie, A. Krüger, P. Olivier, and R. Therón, editors, *Proceedings of the 9th International Symposium on Smart Graphics (SG'09)*, volume 5531 of *Lecture Notes in Computer Science*, pages 43–55. 2009.

[1] *Author's note:* When contacted, M. Sharir kindly sent me what files he still had of the manuscript. Unfortunately, this excludes the figures. An overview of the technique, with some of the details omitted, can be found in [64].

[72] S. D. Peterson, M. Axholt, M. Cooper, and S. R. Ellis. Visual clutter management in augmented reality: Effects of three label separation methods on spatial judgments. In K. Kiyokawa, editor, *Proceedings of the 4th IEEE Symposium on 3D User Interfaces (3DUI'09)*, pages 111–118. 2009.

[73] S. D. Peterson, M. Axholt, M. Cooper, and S. R. Ellis. Detection thresholds for label motion in visually cluttered displays. In B. Lok, G. Klinker, and R. Nakatsu, editors, *Proceedings of the 4th IEEE Virtual Reality Conference (VR'10)*, pages 203–206. 2010.

[74] S. D. Peterson, M. Axholt, and S. R. Ellis. Label segregation by remapping stereoscopic depth in far-field augmented reality. In M. A. Livingston, O. Bimber, and H. Saito, editors, *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR'08)*, pages 143–152. 2008.

[75] S. D. Peterson, M. Axholt, and S. R. Ellis. Objective and subjective assessment of stereoscopically separated labels in augmented reality. *Computers & Graphics*, 33(1), 2009.

[76] I. Petzold, G. Gröger, and L. Plümer. Fast screen map labeling — datastructures and algorithms. In *Proceedings of the 23rd International Cartographic Conference (ICC'03)*, pages 288–298. 2003.

[77] I. Petzold, L. Plümer, and M. Heber. Label placement for dynamically generated screen maps. In *Proceedings of the 19th International Cartographic Conference (ICC'99)*, pages 893–903. 1999.

[78] S.-H. Poon and C.-S. Shin. Adaptive zooming in point set labeling. In M. Liśkiewicz and R. Reischuk, editors, *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT'05)*, volume 3623 of *Lecture Notes in Computer Science*, pages 233–244. Springer-Verlag, 2005.

[79] S.-H. Poon, C.-S. Shin, T. Strijk, T. Uno, and A. Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.

[80] Z. Qin and B. Zhu. A factor-2 approximation for labeling points with maximum sliding labels. In M. Penttonen and E. M. Schmidt, editors, *Proceedings of the 8th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'02)*, volume 2368 of *Lecture Notes in Computer Science*, pages 100–109. 2002.

[81] G. Raidl. A genetic algorithm for labeling point features. In *Proceedings of the 1998 International Conference on Imaging Science, Systems, and Technology (CISST'98)*, pages 189–196. 1998.

[82] E. Rose, D. Breen, K. H. Ahlers, C. Crampton, M. Tuceryan, R. Whitaker, and D. Greer. Annotating real-world objects using augmented reality. In R. A. Earnshaw and J. A. Vince, editors, *Computer Graphics: Developments in Virtual Environments*, pages 357–370. Academic Press, 1995.

[83] E. Rosten, G. Reitmayr, and T. Drummond. Real-time video annotations for augmented reality. In G. Bebis, R. Boyle, D. Koracin, and B. Parvin, editors, *Proceedings of the 1st International Symposium on Advances in Visual Computing (ISVC'05)*, volume 3804 of *Lecture Notes in Computer Science*, pages 294–302. Springer Heidelberg, 2005.

[84] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974.

[85] M. Salganicoff, G. Metta, A. Oddera, and G. Sandini. A vision-based learning method for pushing manipulation. In *Proceedings of the AAAI Fall Symposium on Intelligent Machine Learning in Vision: What Why and How?* 1993.

[86] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[87] M. Schreyer and G. R. Raidl. Letting ants labeling point features. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC'02)*, pages 1564–1569. 2002.

[88] G. Stadler, T. Steiner, and J. Beiglbock. A practical map labeling algorithm utilizing morphological image processing and force-directed methods. *Cartography and Geographic Information Science*, 33(9):207–215, 2006.

[89] T. Stein and X. Décoret. Dynamic label placement for improved interactive exploration. In *Proceedings of the 6th International Symposium on Non-photorealistic animation and rendering (NPAR'08)*, pages 15–21. ACM, 2008.

[90] M. Vaaraniemi, M. Treib, and R. Westermann. Temporally coherent real-time labeling of dynamic scenes. In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications (COM.Geo'12), article no. 17.* 2012.

[91] O. V. Verner, R. L. Wainwright, and D. A. Schoenefeld. Placing text labels on maps and diagrams using genetic algorithms with masking. *INFORMS Journal on Computing*, 9(3):266–275, 1997.

[92] F. Wagner and A. Wolff. A practical map labeling algorithm. *Computational Geometry: Theory and Applications*, 7(5–6):387–404, 1997.

[93] S. Walker and J. K. Salisbury. Pushing using learned manipulation maps. In *Proceedings of the 2008 IEEE International Conference on Robotics & Automation (ICRA'08)*, pages 3808–3813. 2008.

[94] A. Wolff and T. Strijk. The Map Labeling Bibliography. `http://liinwww.ira.uka.de/bibliography/Theory/map.labeling.html`, 2009.

[95] M. Yamamoto, G. Camara, and L. A. N. Lorena. Tabu search heuristic for point-feature cartographic label placement. *GeoInformatica*, 6(1):77–90, 2002.

[96] T. Yoshikawa and M. Kurisu. Identification of the center of friction from pushing an object by a mobile robot. In *Proceedings of the 4th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'91)*, pages 449–454. 1991.

[97] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt. Two-level rrt planning for robotic push manipulation. In *Proceedings of the 25th IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS'12)*, pages 678–685. 2012.

[98] S. Zoraster. Integer programming applied to the map label placement problem. *Cartographica*, 23(3):16–27, 1986.

[99] S. Zoraster. The solution of large 0–1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.

Summary

Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points

This thesis covers two classes of problems that seem unrelated at the surface. The first involves path planning, which is a fundamental problem in robotics where the objective is to find a way to move the robot to its destination, without bumping into obstacles along the way. Manipulation path planning is a variant of this problem, in which the robot is to move a passive object to a certain destination, rather than move itself there. This can be effected in various ways, such as carrying the object, rolling it, or even throwing it. We have studied the problem of pushing a disk-shaped object through an environment of polygonal obstacles using a disk-shaped robot. An existing algorithm by Nieuwenhuisen et al. solves this problem, but their method has several deficiencies. Firstly, their method maintains contact between the robot and the object at all times. We show that there are environments in which the only way to get the object to its destination is to occasionally let go of the object, and then resume pushing from a different angle. Secondly, their method uses a great deal of storage space and computation time to preprocess the obstacles. Lastly, no attempt is made to minimize the effort of the pusher robot to achieve the intended object motion. We have developed a new method that fixes these deficiencies: it is efficient in storage and computation time, can compute the shortest possible push plans in case the robot and object can maintain contact at all times, and can still compute a (not necessarily shortest) push plan in the more complicated case where the robot needs to let go of the object on occasion.

The second problem class we study involves map labeling, which is the task of placing textual labels for features on a map such as cities (points), roads (polygonal paths), and lakes (polygons). For readability, labels should not overlap, which means that one should carefully choose which labels to place and where to place them. It takes cartographers considerable time to do this by hand, and for some applications it is not even possible to do by hand ahead of time. In air-traffic control, for example, a set of moving points (airplanes) has to be labeled at all times, each pulling its label along behind it. In inter-

active maps users may pan, rotate, and/or zoom their view of the map, which may also require relabeling. While there is a vast body of literature on algorithmic methods for the kind of static label placement needed in cartography, far fewer results are known for the kind of dynamic label placement problems inherent in air traffic control and interactive maps. In particular, no study had been made of the theoretical complexity of dynamic labeling. We prove the PSPACE-completeness of a variety of dynamic labeling problems, whose static analogues are NP-complete. In addition, we present a heuristic algorithm—and its experimental evaluation—for labeling a set of moving points. Here the trajectories of the points are assumed to be known beforehand, at least for the immediate future. This has applications in air traffic control (where pilots make their intended course known), and navigation systems (where the driver will presumably follow the computed route). Along the way of developing this dynamic labeling algorithm we identify and fill a gap in the static labeling literature. Labeling a maximal subset of points without overlap is well-studied, but labeling all points while minimizing overlap had not been studied at all. We develop efficient approximation algorithms for this static labeling problem, which are used as a subroutine in our dynamic labeling algorithm.

While static labeling is certainly very different from robot motion planning, when considering dynamic labeling the connection is easily made. Instead of the labels being pulled by their points, we can interpret them as pushing the points along. Labeling moving points then becomes a multi-robot variant of the pushing problem.

Curriculum Vitae

Dirk was born in 1983, and fell in love with programming as soon as he got access to a computer. While implementing additional modes of play for the video games he enjoyed with his friends, he read many books and websites on the practice of programming. Most, he felt, placed too much emphasis on the irrelevant, and lacked a solid theoretical underpinning. To make up for this, he enrolled into the Computer Science program of the Eindhoven University of Technology, in 2002. After attaining his Bachelor and Master's degrees, both *cum laude*, he was asked in 2008 to join the Algorithms group as a PhD student. There he remains as a researcher to this date, combining Computer Science theory and practice with great enjoyment.

Titles in the IPA Dissertation Series since 2007

H.A. de Jong. *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems.* Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains.* Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use.* Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems.* Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML.* Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery.* Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding.* Faculty

of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks.* Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data.* Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines.* Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras.* Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit.* Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks.* Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments.* Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of*

Aspects. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design.* Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems.* Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms.* Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation.* Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems.* Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification.* Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development.* Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation.* Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean.* Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques.* Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time.* Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation.* Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML.* Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange.* Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory.* Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis.* Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks.* Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement.* Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers.* Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics.* Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics.* Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software.* Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra.* Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications.* Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services.* Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols.* Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates.* Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement.* Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components.* Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors.* Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory.* Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution.* Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes.* Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains.* Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations.* Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice.* Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars.* Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns.* Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems.* Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited.* Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software.* Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective.* Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development.* Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination.* Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain.* Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems.* Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards.* Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor.* Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins.* Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking.* Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points.* Faculty of Mathematics and Computer Science, TU/e. 2013-12

Stellingen behorend bij het proefschrift

Pushing and Pulling
Computing push plans for disk-shaped robots,
and dynamic labelings for moving points

van Dirk Gerrits

1. Approximation algorithms for number maximization are of no use to a cartographer trying to label cities on a map.

2. The definition of weighted number maximization for slider models in the literature does not properly generalize the one for fixed-position models.

3. Min Voronoi Connectivity [1] is defined as follows: given sets $\mathcal{B}$ and $\mathcal{R}$ of points in $\mathbb{R}^d$, compute the minimum-cardinality $\mathcal{B}_{\text{del}} \subseteq \mathcal{B}$ such that $\mathcal{R}$ induces a connected subgraph of the Delaunay graph of $\mathcal{R} \cup \mathcal{B} \setminus \mathcal{B}_{\text{del}}$.

   This problem is NP-hard in general, but can be solved optimally in polynomial time if $|\mathcal{B}|$ is constant, if $|\mathcal{R}| = 2$, or if both $|\mathcal{R}|$ and d are constant. Additionally, one can obtain factor $O(|\mathcal{R}|)$ and $O(|\mathcal{B}|)$ approximations in polynomial time.

4. As crucial a tool as visualization has become to the sciences, so under- and misused it is in the humanities.

5. To truly understand an algorithm, one has to implement it.

   Corollary: No one truly understands Chazelle's linear-time triangulation algorithm [3].

6. The age of the time-space tradeoff is over.

7. The phrases "unpublished manuscript" and "personal communication" have no place in the reference list of a scientific publication.

8. The phrase "it is easy to see that" is a wonderful tool. One admits not being able to find an adequate proof, while making the *reader* feel stupid.

9. The Dutch policy for research funding is homeopathic.

   The more dilute the funding, it is believed, the greater its efficacy. That any research still gets done can only be ascribed to the placebo effect.

10. Open Access should have become the default years ago.

    The public has a right to the research it helped to fund. Yet, of journal articles published in 2008, only 1 in 5 can be freely accessed online, even counting those available from their author's website [2].

11. Trying to save money by cutting on the mental health care budget can only have the opposite effect.

    No ailment causes more person-years to be lost to disability than depression [4]. Stigma alone is enough of a barrier for seeking help; money should not be another.

12. Stacks make poor priority queues.

    As such, getting out of a crowded elevator on anything but its final floor can be an inefficient ordeal.

13. A picture's value cannot be expressed in words.

14. Everyone is born a scientist; some parents just handle their child's "Why?"-phase poorly.

References

[1] M. de Berg, D. H. P. Gerrits, A. Khosravi, I. Rutter, C. P. Tsirogiannis, and A. Wolff. How Alexander the Great brought the Greeks together while inflicting minimal damage to the barbarians. In *Proceedings of the 26th European Workshop on Computational Geometry (EuroCG'10)*, pages 73–76. 2010.

[2] B.-C. Björk, P. Welling, M. Laakso, P. Majlender, T. Hedlund, and G. Guðnason. Open access to the scientific journal literature: Situation 2009. *PLoS One*, 5(6):e11273, 2010.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.

[4] M. Marcus, M. T. Yasamy, M. van Ommeren, D. Chisholm, and S. Saxena. Depression: A global public health concern. Department of Mental Health and Substance Abuse, World Health Organization, 2012.