Open access • Proceedings Article • DOI:10.1145/109648.109680

A packing problem with applications to lettering of maps — Source link

Michael Formann, Frank Wagner

Institutions: Free University of Berlin

Published on: 01 Jun 1991 - Symposium on Computational Geometry

Topics: Set packing, Cutting stock problem, Rectangle, Packing problems and Approximation algorithm

A packing problem with applications to lettering of maps*

MICHAEL FORMANN AND FRANK WAGNER
Institut für Informatik, Fachbereich Mathematik, Freie Universität Berlin,
Arnimallee 2–6, W1000 Berlin 33, Germany
e-mail: formann@tcs.fu-berlin.de, wagner@tcs.fu-berlin.de

Abstract

The following packing problem arises in connection with lettering of maps: Given n distinct points $p_1, p_2, \ldots, p_n$ in the plane, determine the supremum σ_{opt} of all reals σ, such that there are n pairwise disjoint, axis-parallel, closed squares $Q_1, Q_2, \ldots, Q_n$ of side-length σ, where each p_i is a corner of Q_i. Note that — by using affine transformation — the problem is equivalent to the case when we want largest homothetic copies of a fixed rectangle or parallelogram instead of equally-sized squares.

In the cartographic application, the points are items (groundwater-drillholes etc.) and the squares are places for labels associated with these items (sulphate concentration etc.).

An algorithm is presented, that in $\mathcal{O}(n \log n)$ time either produces a solution, that is guaranteed to be at least half as large as the supremum. This is optimal, in the sense that the corresponding decision problem is $\mathcal{NP}$-complete, no polynomial approximation algorithm with a guaranteed factor exceeding $\frac{1}{2}$ exists, provided that $\mathcal{P} \neq \mathcal{NP}$; and there is also a lower bound of $\Omega(n \log n)$ for the running time.

1 Introduction

The more there is a need for large, especially technical maps, for which legibility is much more important than beauty, the more the computerization of maps asks for fully automated algorithms.

A basic task in the process of producing maps is the *lettering*, the positioning of labels which describe properties of points on the map, such that the inscriptions are legible, i.e. large enough and non-overlapping, and a user of the map can easily and intuitively identify the label of a specific point.

An adequate formalization is the following:

Problem R4 (Rectangles in four positions) We are given n distinct points in the plane each associated with an axis-parallel rectangle and we want to know whether it is possible to shift these rectangles horizontally and vertically such that

(i) all rectangles are pairwise disjoint and

(ii) each point is a corner of "its" rectangle.

If the answer is positive, of course we also want to know the solution – the position of the rectangles.

Note, that each rectangle can be placed exactly in four possible positions by restriction (ii). Figure 1 shows an example for seven points, the four possible positions of each rectangle are drawn with thin lines, a solution is indicated in bold.

Cartographers often get into trouble with the lettering of maps, in particular naïve computer-based approaches produce either overlappings of inscriptions or, in order to avoid that, omit some of the labels, thus producing illegible or incomplete maps. Incompleteness is not that large a problem in usual geographical maps, where the points are cities and the labels are their respective names; one may leave out some small or less important citites. But in technical maps, there is usually no unimportant information.

Cartographers in the municipal authorities of München encountered the problem for the simpler case of equally-sized rectangles, when they tried to automate

*This work was partially supported by the ESPRIT II Basic Research Action of the European Community under contract Nos. 3075 and 3299 (project ALCOM and working group "Computing by Graph Transformations").

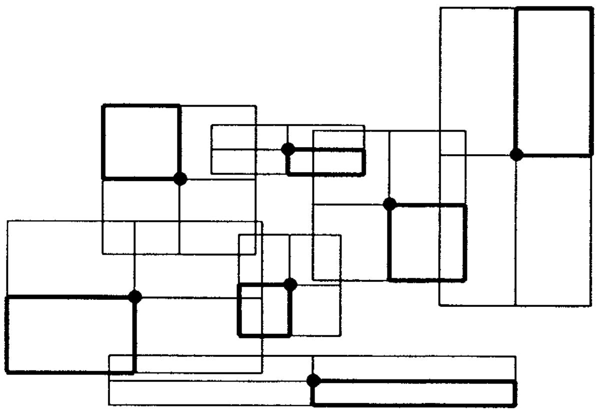


Figure 1: An example for 7 points.

the production of labellings for groundwater maps. There the points are drill-holes and the rectangles are associated labels, e.g. the name of the drill-hole, the groundwater level, the sulphate concentration etc.

Unfortunately, problem R4 is $\mathcal{NP}$-complete, even for equally-sized squares. This will be shown at the end of this paper in section 5. Firstly in section 2 we will show, how to solve efficiently a variant of the problem, where we allow only two fixed out of the four possible positions for each rectangle. This partial result is not of great interest in itself, but it is used as the key subroutine of the main algorithm, which comes in section 3, where we tackle the original problem from another point of view. We start with infinitesimal equally-sized squares and want to know how big we can simultaneously "blow up" the squares, such that we still have a solution. An algorithm will be presented, that blows up the squares to at least 50% of the maximal size. We also show, provided that $\mathcal{P} \neq \mathcal{NP}$, no better approximating algorithm with polynomial runtime exists. Our algorithm runs in $\mathcal{O}(n \log n)$ optimal time.

2 Two positions

As indicated in the introduction, we can solve the problem efficiently if we allow only two out of the four possible positions for each rectangle, in the sequel we will call this problem R2 (Rectangles in two positions).

The algorithm is fairly simple and proceeds in three steps.

Algorithm AR2 (Algorithm for rectangles in two positions)

1. We consider the points as Boolean variables. Denominate the two positions of the rectangle associated with the i^{th} point by x_i and $\overline{x_i}$.
2. If two placements overlap, say x_i and $\overline{x_j}$, form the clause
$$\overline{(x_i \wedge \overline{x_j})} = \overline{x_i} \vee x_j.$$
This gives us a set of clauses.
One should interprete the left side of the equation as: "We *don't* want, that x_i *and* $\overline{x_j}$ are simultanously in a solution, because these positions overlap".
3. Check, if there is a satisfying truth assignment for the set of clauses. If there is such an assignment take position x_i ($\overline{x_i}$) for the i^{th} rectangle, if variable x_i is set to the value true (false) in this assignment.

Now it is obvious, that a satisfying truth assignment gives us a valid (non-overlapping) solution for R2 and vice versa.

Step (3) of algorithm AR2 is simply the classical 2-SAT problem, which was shown to be solvable in time linear in the number of clauses by [Even, Itai and Shamir]. So, what is the number of clauses and how fast can we generate the clauses?

In general, we can have $\Omega(n^2)$ pairwise intersections of rectangles and therefore $\Omega(n^2)$ clauses. Note, that there are instances of R2 with $\Theta(n^2)$ clauses, that still have solutions (cf. Figure 3). [Imai and Asano] exploit the geometric flavour of the problem and present an $\mathcal{O}(n \log^2 n)$ algorithm for Problem R2. Still, one can do better for equally-sized squares, since then there are at most $\mathcal{O}(n)$ intersections in solvable cases.

Lemma 1 *Given an instance of R2 with equally-sized squares. If any square is cut by more than 24 other squares than there exists no valid solution.*

Proof: Let Q be any square. W.l.o.g. the lower left corner p of Q has coordinates (0,0). Then any square that cuts Q must lie within the square spanned by the points (-1,-1) and (2,2). Furthermore all their "partner squares" (the other possible positions of the cutting squares) must lie within the square spanned by (-2,-2) and (3,3). (see Figure 2).
This big square has an area of 25 square units and we have to choose one square from each pair (square that cuts s, its partner). If more than 24 squares cut s, any partial solution for these pairs and for s has to use more than 25 square units, a contradiction. ⌸

p intersections can be found in $\mathcal{O}(n \log n + p)$ time [Edelsbrunner]. Therefore we can sum up the results of this chapter:

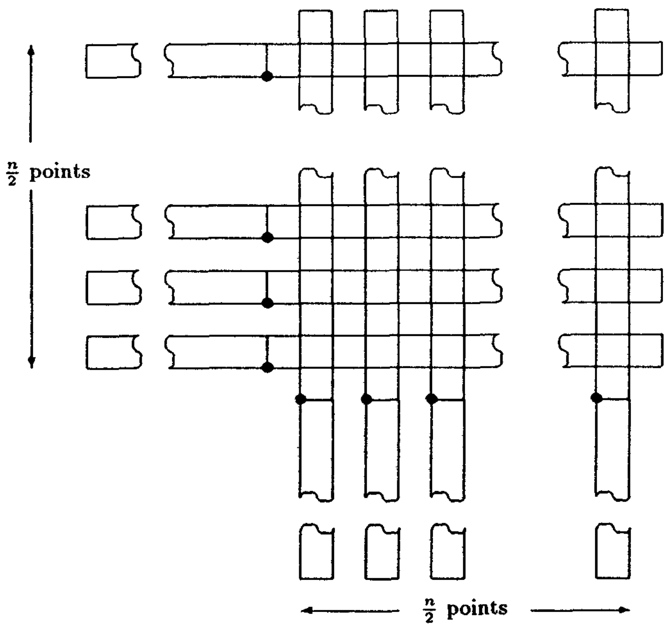


Figure 3: A solvable example with $\Theta(n^2)$ squares.

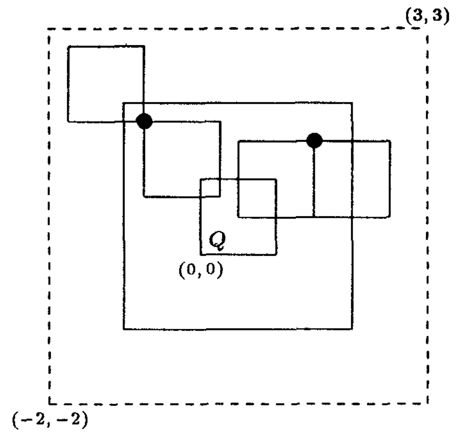


Figure 2: Any square that cuts s or whose partner cuts s must lie within the dashed big square.

Theorem 2 *Problem R2 is solvable in $\mathcal{O}(n \log^2 n)$ time for arbitrary rectangles and in $\mathcal{O}(n \log n)$ time for equally-sized squares.*

3 Approximating the optimal size

Apart from the fact that – most likely – we cannot solve the problem R4 in polynomial time; even if we could, a negative answer would not be very helpful, because we must label our points somehow by all means. So in this case we should better use a larger scale or, in other words, smaller labels.

In this section we therefore consider the following variant of our original problem.

Problem S4a (Squares in four positions, approximation) Given n distinct points in the plane, what is the supremum σ_{opt} of all reals σ, such that there is a closed square with side length σ for every point with the two properties:

(i) the point is in one of its corners and

(ii) all squares are pairwise disjoint.

σ_{opt} will be called the *optimal size*.

Note that, it makes a difference whether we consider the rectangles as being topologically closed or open. Take four equidistant points on a horizontal line. In the first case the optimal size is just the distance between two points but in the second case the solution is infinitely large. For problem R2 in the previous section we did not state explicitly, which of the two cases we meant, because our methods work for both of them.

Here, this is not the case. The key fact (Lemma 3), which enables us to develop an approximation algorithm, does not hold for topologically open rectangles.

With topologically open rectangles it can be the case, that from a labelling we cannot reconstruct uniquely the point which belongs to a label. This might not be very useful for practical purposes.

As already said, the best we can hope for is an algorithm that approximates α by at least 50%. This will be proved in section 5.

Intuitively our algorithm works along the following lines:
We start with infinitesimal equally-sized squares and want to know how big we can simultaneously "blow up" the squares, such that we still have a solution. The inherent difficulty of our problem is that we have to choose one of 4^n possible solutions, since each of the four positions of each point is a candidate in the beginning. With growing current size σ we exclude as much squares as possible from being a candidate by eliminating them, thus reducing the number of possible solutions.

Of course, we have to be sure that we do not eliminate squares which are needed in a solution of the size we aim at (at least 50% of the optimal size).

We divide the squares into three different types for a fixed current size σ:

Definition 1 *For a point p in the plane denote by p_i ($i \in \{1,2,3,4\}$) an axis-parallel unit square with p in its southwest, southeast, northeast resp. northwest corner (the numbering is chosen like that of quadrants); for a real σ denote by σp_i analogously a square with side-length σ.*
For a point set P and a real σ we call p_i ($p \in P$)

1. σ-dead, *if $2\sigma p_i$ contains another point $q \in P$;*
2. σ-pending, *if p_i is not σ-dead and σp_i has nonempty intersection with σq_j, where q_j is another non-σ-dead square;*
3. σ-alive, *if p_i is neither σ-dead nor σ-pending.*

The algorithm can be formulated very simply now. It is designed in order to maintain two invariants: Firstly, only σ-dead squares are eliminated and secondly it can always efficiently construct a solution of the current size from the set of those squares, which are not yet eliminated.

By using AR2 in step 2.2 of the following algorithm we assume, that the case that three or more of a vertices squares are pending, cannot occur. This assumption will be justified by Lemma 3 below.

Algorithm AS4a (algorithm for squares in four positions, approximation)

```
1 Start with all four squares of size $\sigma := 0$ assigned
  to each point;
2 Let $\sigma$ grow:
2.1       eliminate all $\sigma$-dead squares;
2.2       for the set of all points that have no $\sigma$-alive
          square use the algorithm AR2;
2.2       stop if one point has no more square or the
          algorithm AR2 in step 2.2 gives the
          answer "no";
3 decrease the stop-value of $\sigma$ by an arbitrary small
  amount resulting in $\sigma_{res}$.
4 Construct a solution:
4.1       for every point that still has $\sigma_{res}$-alive
          squares, choose one of them;
4.2       for the rest take the solution of AR2;
```

Note, that it is possible that the loop in the algorithm does not stop as σ tends to infinity. This is the case, if there are at most four points, all of which are vertices of the convex hull. We exclude this special case for the rest of the discussion.

The first invariant, described above, obviously holds. For this reason we can conclude, that we do not eliminate squares which are needed in a solution of the size we aim at, this can be seen as follows:
From a solution of optimal size σ_{opt} we can obviously construct an approximate solution of the claimed quality by shrinking the squares to half of their size. No such half-size square is eliminated as long as the current size is at most $\sigma_{opt}/2$. This is the case because these squares are not $\sigma_{opt}/2$-dead.

The second invariant we claimed, is that, as long as the stop-condition is not fulfilled, we can always efficiently construct a solution of the current size from the set of those squares, which are not yet eliminated. To see this, note that a σ-alive square does not intersect with any other square of size σ, that is not yet eliminated. So for all points, that still have σ-alive squares we are able to take one for a solution. For the other points we know by the first part of the stop-condition, that each of them has at least one σ-pending square. The second part of the stop-condition guarantees, that we can complete the solution according to the truth assignment given by algorithm AR2.

The algorithm stops, when it reaches a size where no such solution exists anymore. But, what will be the output? Of course we have stored the status of the squares at the size, which was current just before the stop-condition was fulfilled!

As already stated implicitely we assume in the algorithm, that the case that three or more of a vertices squares are pending, cannot occur. This assumption will be justified by the following Lemma 3. It is somehow the most important tool in our algorithm because by this lemma we can use the polynomial-time 2-SAT algorithm.

Lemma 3 *The number of σ-pending squares of a point is at most two.*

Proof: Suppose p_1 is σ-pending. Then there exists another square q_j such that σp_1 and σq_j have nonempty intersection. But none of them is σ-dead by definition, so neither $2\sigma p_1$ ($2\sigma q_j$) contain another point, especially not q resp. p. But then q must lie in the square σp_2 or σp_4; note that $j = 1$ and $q_j = q_1$ must be σ-pending too. It follows that σp_2 or σp_4 is then certainly σ-dead (see Fig. 4).

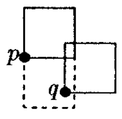


Figure 4: illustration for the proof of Lemma 2

Analogous facts hold for p_2, p_3 and p_4. Now suppose three squares of p, w.l.o.g. p_1, p_2 and p_3 are pending. All corresponding "pending partners" must lie in σ-dead squares of p. This can be managed for p_1 and p_3 but not for p_2. ⊡

To summarize:

Theorem 4 *Algorithm AS4a constructs a solution for problem S4a with at least 50% of the optimal size.*

Clearly, be rescaling one of the coordinates, our algorithm can also be applied to equally-sized rectangles. Furthermore, by using affine transformations, our algorithm can be used for such strange label shapes as parallelograms.

4 Analysis of the algorithm

In this chapter we will sketch how the algorithm AS4a can be implemented efficiently.

The main difficulty lies in finding those values of σ, where a square changes its status. Firstly, it is clear that a square becomes σ-dead only once. The $4n$ values of σ where this happens, can be found by four plane-sweeps (one for each of the four square types) in $\mathcal{O}(n \log n)$ total time.

For σ-pending squares the situation is more complicated. Note, that by definition a σ-pending square has non-empty intersection with *at least* one other (σ-pending) square. In the next Lemma we will show, that the number of those "pending partners" cannot be to large.

Lemma 5 *Only the first 13 values of σ when a square gets another pending partner must be considered in algorithm AS4a for each square.*

Proof: By a similar packing argument as in the proof of Lemma 1. ⊡

These 13 values of σ can be found by plane-sweep techniques again in $\mathcal{O}(n \log n)$ time. So, summing up we can find all $\mathcal{O}(n)$ relevant σ-values (events), where the status of a square changes, in $\mathcal{O}(n \log n)$ time.

By Lemma 5 the total number of clauses ever considered in step 2.2 by using AR2 as subroutine is linear. How can we circumvent the difficulty that one single satisfiability test will cost us $\mathcal{O}(n)$ time and we have to do $\mathcal{O}(n)$ of these, resulting in $\mathcal{O}(n^2)$ running time? We do this by ignoring line 2.2 of the algorithm and running the algorithm till the events are used up. Then, by binary search, we find the value when in the original version our algorithm would have stopped, because the set of clauses became unsatisfiable. This results in the following Lemma.

Lemma 6 *Algorithm AS4a can be implemented to run in $\mathcal{O}(n \log n)$ time.*

In the full paper we will show by using [Ben-Or]'s methods that $\Omega(n \log n)$ is a lower bound as well for the "exact problem" R4 as for the approximation problem S4a.

5 $\mathcal{NP}$-completeness result

As already stated, problem R4, is $\mathcal{NP}$-complete. We will show this for a very special case, the proof will only be sketched.

Theorem 7 *Problem R4 is $\mathcal{NP}$-complete even if all rectangles are squares of the same size.*

Proof: Obviously, the problem is in $\mathcal{NP}$. We show the $\mathcal{NP}$-hardness by reducing 3-SAT (cf. [Garey and Johnson], p. 259) to it. Recall that 3-SAT is the following problem:

> 3-SAT (3-SATISFIABILITY)
> INSTANCE: Set U of variables, collection C of clauses over U, such that each clause has exactly three literals.
> QUESTION: Is there a satisfying truth assignment for C?

For each variable we use one special point, whose square is forced – by other points – to be in two of the four positions, below left or below right. The position taken symbolizes the setting of the variable, true or false. From each variable point x "pipes" lead to the clauses that contain x or $\overline{x}$. For negated variables the pipes are flanged at the "true-side" of the variables and unnegated on the "false-side". The idea is that the pipes transmit the "pressure" produced by the setting of the variable squares. Finally the pipes lead into the clauses. These clauses are constructed in such a way that they can stand pressure from at most two pipes. Recall that these pipes carry a variable setting that does not satisfy the clause. So, either each clause takes up pressure by at most two non-satisfying pipes and is therefore satisfied or there is a clause which is expected to take up pressure from three pipes. Then there is no solution for problem R4. So, there is a solution for the given instance of 3-SAT if and only if problem R4 has a solution for the constructed point set. Figure 5 gives the basic construction and Figure 6 shows how the details are modelled. ⊔⊓

Lemma 8 *Provided that $\mathcal{P} \neq \mathcal{NP}$, no polynomial-time algorithm with a guaranteed factor exceeding $\frac{1}{2}$ exists.*

Proof: Just look at the detailed construction of our $\mathcal{NP}$-completeness proof in figure 6.

If the set of clauses is satisfiable, then there exists also a solution with squares nearly twice as large as the unit-squares in our construction. Note, that in the figure all squares can be blown up to twice their size simultanously without intersection.

So an algorithm with better approximation-quality would produce a solution with bigger squares than our unit-squares.

On the other hand, if the set of clauses is not satisfiable such an algorithm would produce a solution with small squares (smaller than unit-size).

So we would have a polynomial-time decision algorithm for problem R4. ⊔⊓

Acknowledgement

We would like to thank Rudi Krämer (Amt für Informations- und Datenverarbeitung der Stadt München) and Kurt Mehlhorn for bringing this problem to our attention. We would also like to thank Emo Welzl for a simplification of the presentation of algorithm AS4a.

References

[Ben-Or] M. Ben-Or, *Lower bounds for algebraic computation trees* Proc. 15th ACM Ann. Symp. Theory of Computing (1983), 80-86

[Edelsbrunner] H. Edelsbrunner, *Dynamic data structures for orthogonal intersection queries*, Rep. F59, Technische Universität Graz, Institute für Informationsverarbeitung (1980)

[Even, Itai and Shamir] S. Even, A. Itai and A. Shamir, *On the complexity of timetable and multicommodity flow problems*, SIAM J. Comput. **5** (1976), 691-703

[Garey and Johnson] M. R. Garey and D. S. Johnson, *Computers and Intractability, A guide to the theory of NP-completeness*, Freeman, 1979

[Imai and Asano] H. Imai and T. Asano, *Efficient Algorithms for Geometric Graph Search Problems*, SIAM J. Comput. **15** (1986), 478-494

[Imhof] E. Imhof, *Positioning Names on Maps*, The American Cartographer **2** (1975), 128-144

[Yoeli] P. Yoeli, *The Logic of Automated Map Lettering*, The Cartographic Journal **9**, 1972, 99-108

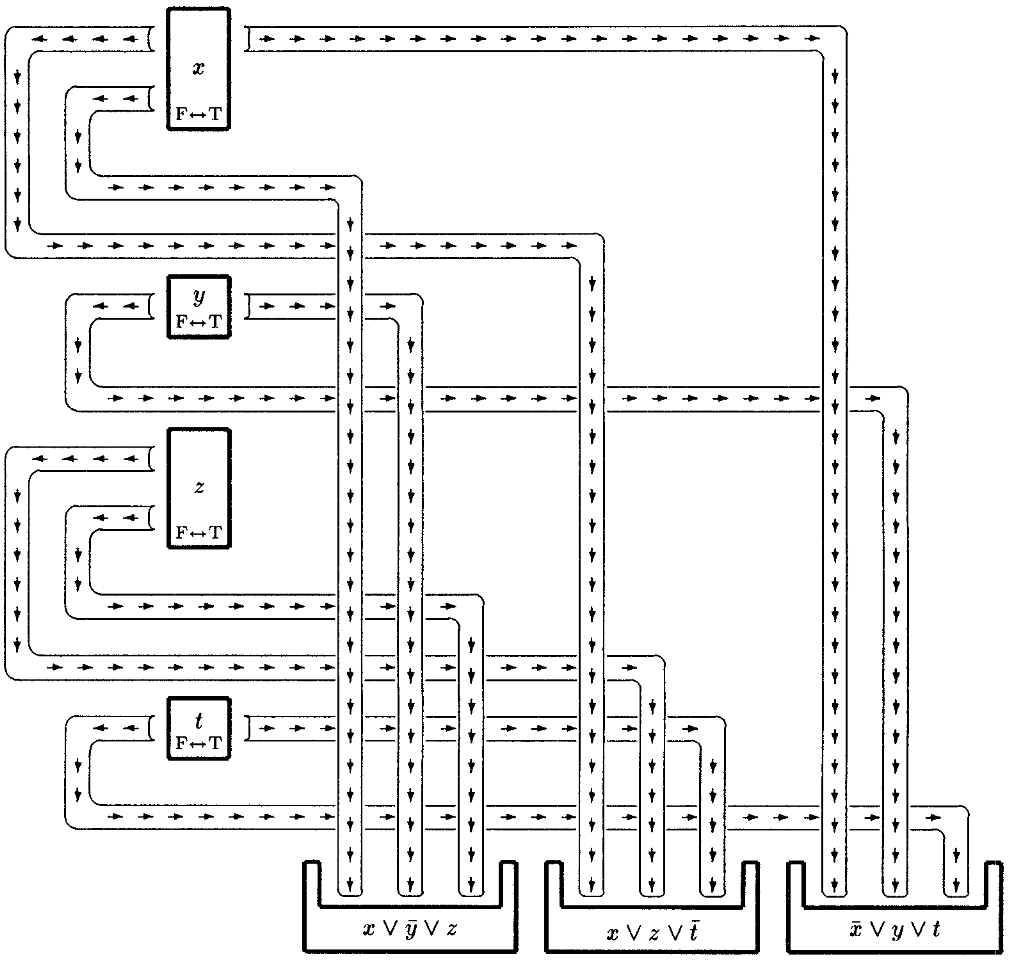
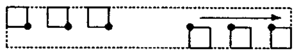
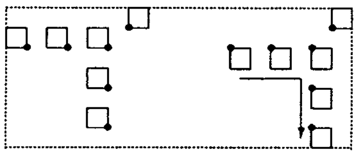
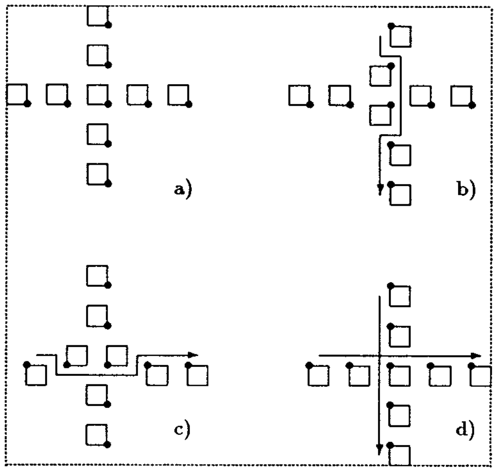


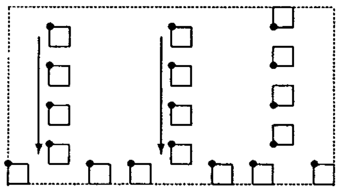
Figure 5: The basic construction; example for the formula $(x \vee \overline{y} \vee z) \wedge (x \vee z \vee \overline{t}) \wedge (\overline{x} \vee y \vee t)$

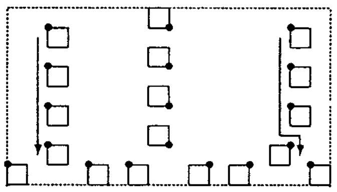

pipe

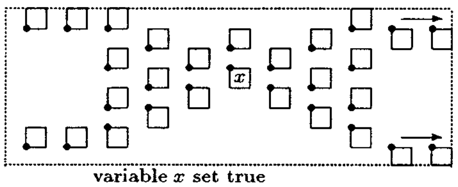

knee of a pipe



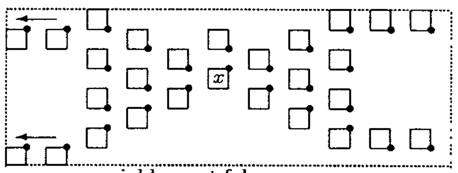
crossing of two pipes, transmitting pressure from a) no, b,c) one and d) both pipes


clause, receiving pressure from two pipes


clause, receiving pressure from two pipes



variable x set true



variable x set false

Figure 6: