Placing Base Stations in Wireless Indoor Communication Networks

Thom Frühwirth, Ludwig-Maximilians-Universität München
Pascal Brisset, Ecole Nationale de l'Aviation Civile

THE POPULAR PROTOTYPE LETS USERS COMPUTE THE MINIMAL NUMBER OF BASE STATIONS AND THEIR LOCATION, GIVEN A BLUEPRINT OF THE INSTALLATION SITE AND INFORMATION ABOUT THE WALL AND CEILING MATERIALS.

MORE AND MORE, MOBILE COMmunications comes to company sites through local, typically indoor, wireless communication networks. No cabling is required, and employees can be reached anytime, anywhere at the site. However, planning wireless networks is quite different from planning traditional wire-based networks.

Planning must take into account the specifics of radio wave propagation at the installation site. Current systems are cellular in that a base station (that is, a sender or transmitter) controls the links to the transceivers. A *radio cell* is the volume that a single base station covers, usually tens of cubic meters. Buildings require multicellular systems because walls and floors absorb part of the radio signal.

Today, an experienced salesperson estimates the number and positioning of base stations. To help the salesperson, Siemens has compiled a set of guidelines based on typical scenarios. However, a scenario might not always apply, and the approach does not handle positioning the base stations well.

Computer-aided planning can help ease the difficulties of planning. Toward that end, we have developed Popular (Planning of Picocellular Radio). Given the building's blueprint and information about the wall and ceiling materials, Popular quickly computes the minimal number of base stations and their location.

Modeling picocellular radio

Radio wave propagation suffers mainly from

- attenuation (weakening) of the signal caused by distance,
- shadowing (absorption) through obstacles,
- multipath propagation caused by reflection and diffraction,
- interference with other base stations, and
- motion in the radio field (the Doppler effect).

Figure 1 shows an example of the resulting path loss over distance on a logarithmic scale. At six and nine meters, walls weaken the signal.

The COST (European *C*ooperation in the Field *o*f *S*cientific and *T*echnical Research) Propagation Models Subgroup proposed this path-loss model:[1]

$$L_P = L_{1\,\mathrm{m}} + 10n\log_{10} d + \sum_i k_i F_i + \sum_j p_j W_j \quad (1)$$

where

- L_P is the total path loss in dB,
- $L_{1\,\mathrm{m}}$ is the path loss one meter from the base station,
- n is the propagation factor,
- d is the distance between the base station and receiver,

1094-7167/00/$10.00 © 2000 IEEE

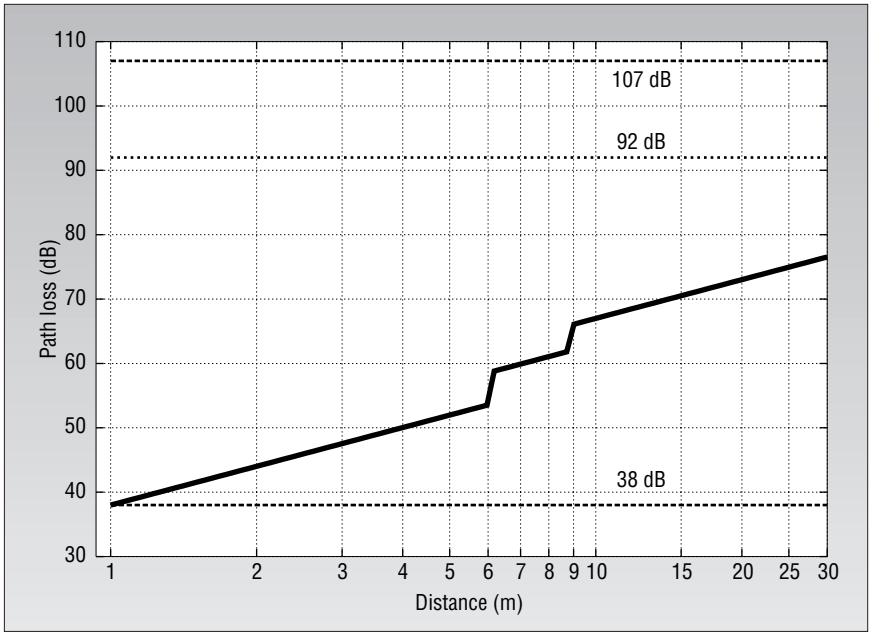

Figure 1. Path loss with additional attenuation caused by walls at six and nine meters.

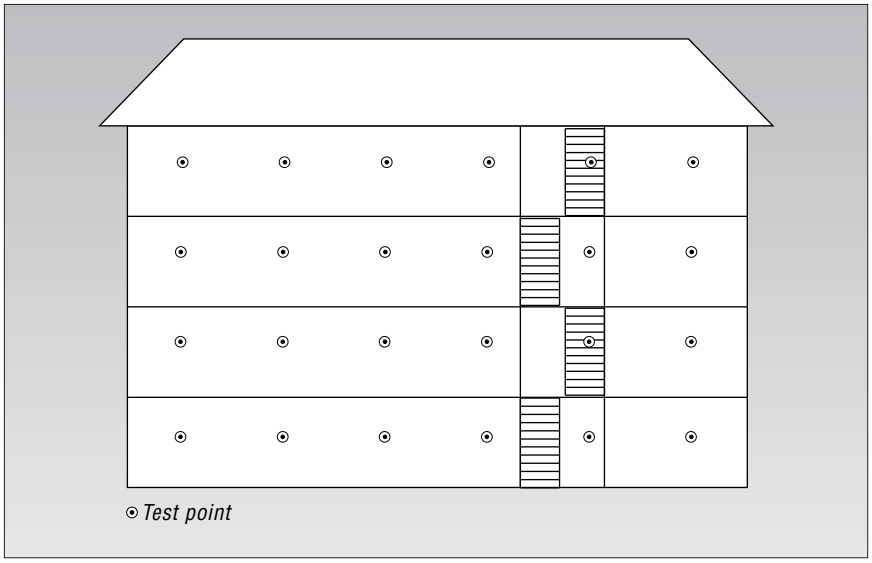

Figure 2. Grid of test points in a building.

- k_i is the number of floors of kind i in the propagation path,
- F_i is the attenuation factor of one floor of kind i,
- p_j is the number of walls of kind j in the propagation path, and
- W_j is the attenuation factor of one wall of kind j.

The model is based on the power balance of wireless transmission. It combines a distance-dependent term with correction factors for extra path loss caused by the building's floors and walls in the propagation path.

However, this path-loss model does not take into account reflection and thus multipath effects. Even with sufficient receiver sensitivity, a radio link could fail because of fading and too many bit errors that result from it. So, we introduced a *fading reserve* (fade margin). We also extended the model to take into account an antenna's directional effect, because antennas do not beam with the same energy in every direction.

Planning in Popular

While working at the European Computer-Industry Research Centre (ECRC) in 1995, we implemented in a few months an initial prototype in the Eclipse constraint-logic programming language.[2] Eclipse includes a library for Constraint Handling Rules,[3] a high-level language extension to implement arbitrary constraint systems. The prototype is part of the demo suite of Eclipse 3.4. Based on this prototype, Jörg-Rainer Molwitz, a student from the University of Aachen, implemented the Popular tool within one man-year while at Siemens.

To give Popular a building's description, the user scans in the blueprint. Popular's interface lets the user redraw walls and ceilings and specify an attenuation factor for each. To compute the minimal number of base stations and their location, Popular first simulates radio wave propagation. It then optimizes the number of base stations needed to cover the whole building.

Radio cell simulation. Popular uses *test points* to compute the building's characteristics. Each test point represents a possible receiver position. Popular places the test points on a 3D grid inside the volume to be covered. Each floor of the building has one such layer of test points (see Figure 2). For each test point, Popular calculates the radio cell that would cover it. If the test grid is sufficiently small (several per square meter), we can expect that if two neighboring test points are covered, the space in between—and hence the whole building—can also be covered.

To simulate the propagation of radio waves through the building's walls and ceilings, Popular uses ray tracing. To get to the point of minimal sensitivity (that is, the maximal permissible path loss), the simulation must follow each path through the whole building (see Figure 3). The maximal permissible path loss comprises the antenna attenuation in the path's direction, the path loss caused by the distance, and the insertion losses caused by intersections of the path with walls and floors. The resulting end points describe the radio cell's hull. To speed up the simulation, we use binary (dichotomy) search to find each ray's threshold location. For each test point, Popular computes 128 rays.

The radio cell will usually be oddly shaped, because the coverage is not a smooth or even differentiable function. The received power at a single point might exhibit discontinuities because of tiny changes in the base station location—for example, a move around a corner can cause an entirely different pattern of transmitted rays. This is why

we cannot express the path-loss formula as a constraint and why we must use ray tracing. The coverage that can be computed is an approximation—that is, limited in accuracy—and small interferences can be neglected. For example, we do not calculate the effect of every piece of furniture.

In practice, the base stations are installed at the same height at each floor, on the ceiling or on the walls. So, on each floor, the possible space of locations for base stations is on a single plane. This plane intersects with the radio cells, reducing them from a polyeder to a series of connected polygons, one for each floor (see Figure 4).

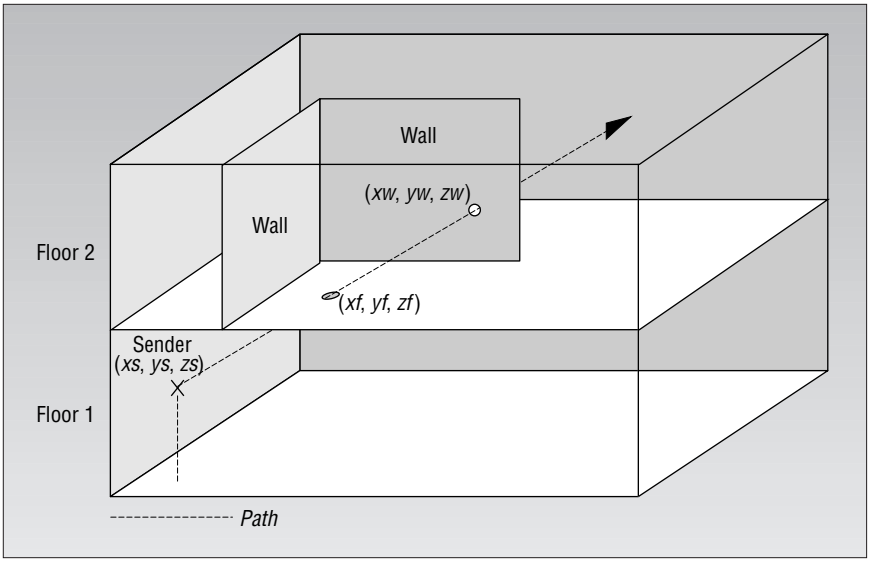

Figure 3. Path from base station intersecting a ceiling and a wall.

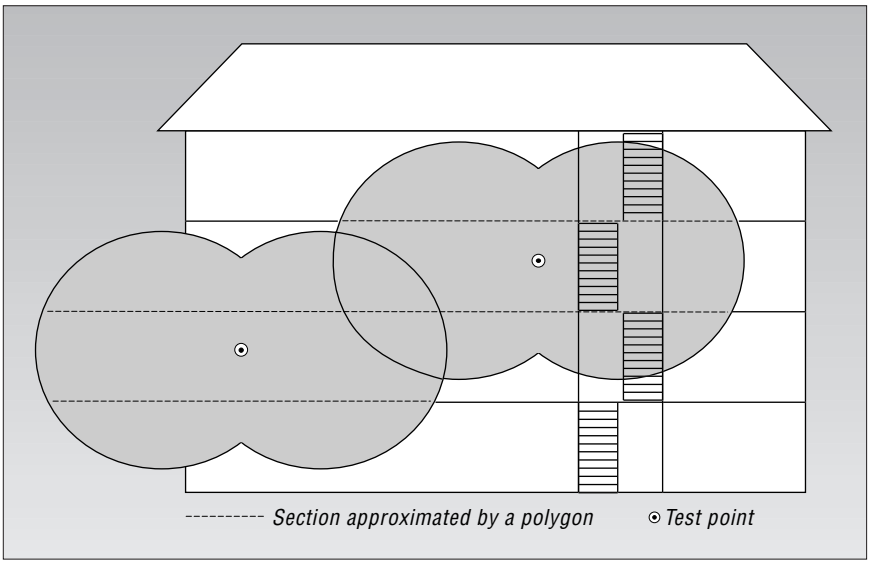

Figure 4. Typical radio coverage areas in a building.

Constraint-based optimization. For each resulting polygon, we impose the constraint that it must contain at least one base station (geometrically speaking, a point). Then, we try to find locations that are in as many polygons as possible at the same time. So, a base station at one of these locations will cover several test points at once. Thus, we constrain the possible locations to be in the intersections of the polygons covered. In this way, we compute a first solution.

Next, to minimize the number of base stations, we use a branch-and-bound method. This method repeatedly searches for a solution with a smaller number of base stations until it finds the minimal number.

To implement the constraint solver, we used Constraint Handling Rules. CHR is essentially a committed-choice language consisting of multiheaded guarded rules that rewrite constraints into simpler ones until they are solved.

In a first attempt restricted to two dimensions, we approximated a polygon by a single rectangle. The 2D coordinates are of the form `X#Y`; rectangles are orthogonal to the coordinate system and are represented by a pair composed of their left-lower- and right-upper-corner coordinates. For each polygon, we simply impose the constraint `inside (Sender, Rectangle)`, where `Sender` refers to a point that must be inside the `Rectangle`.

Figure 5a shows the CHR code for the `inside` constraint. The first rule (named `not_empty`) says that the constraint `inside(S, XL#YL - XR#YU)` is only valid if the condition `XL < XR, YL < YU` is fulfilled, so that the rectangle has a nonempty area. The `intersect` rule says that if a base station's location `S` is constrained by two `inside` constraints to be in two rectangles at once, we can replace these two constraints by a single `inside` constraint whose rectangle is computed as the intersection of the two initial rectangles.

To compute a solution, after we have set up the `inside` constraints, we try to equate as many base stations as possible. (We call this the *labeling phase* of the constraint computation). A simple way to do this is the recursive procedure in Figure 5b. For each base station `S`, the piece of code `(member(S, L) or true)` either nondeterministically equates `S` with one of the remaining base stations in the list `L` using the `member` constraint or does not do so (`true`). Equating base stations causes the `intersect` rule to fire with the constraints associated with the base stations.

This labeling procedure constrains a base station's location more and more, so the `intersect` rule fires again and again until the rectangle becomes very small and finally empty. Then, the `not_empty` rule fires and causes failure, thereby initiating chronological backtracking that will lead to another choice.

This heuristic for equating base stations works efficiently because it takes advantage of the problem's geometric nature and equates base stations from polygons associated with nearby test points first. The optimization produces a few base stations constrained to be inside some small rectangle.

In just 10 minutes, we extended this solver to work with the union of rectangles, which lets it describe the polygon with any desired

```
% inside(Sender, LeftLowerCorner - RightUpperCorner)

not_empty @ inside(S, XL#YL - XR#YU) ==> XL < XR, YL < YU.
intersect @ inside(S, XL1#YL1 - XR1#YU1),
  inside(S, XL2#YL2 - XR2#YU2) <=>
  XL is max(XL1, XL2), YL is max(YL1, YL2),
  XR is min(XR1, XR2), YU is min(YU1, YU2),
  inside(S, XL#YL - XR#YU).
```
(a)

```
equate_senders([]) <=> true.
equate_senders([S|L]) <=>
  (member(S, L) or true), % equate S with another sender or not
  equate_senders(L).
```
(b)

```
intersect @ inside(S, L1), inside(S, L2) <=>
  intersect_geoms(L1, L2, L3),
  L3 = [_|_], % at least one geom left
    inside(S, L3).

intersect_geoms(L1, L2, L3) <=>
  setof(Geom, intersect_geom(L1, L2, Geom), L3).
intersect_geom(L1, L2, XY#YL - XR#YU) <=>
  member(rect(XL1#YL1 - XR1#YU1), L1),
  member(rect(XL2#YL2 - XR2#YU2), L2),
  XL is max(XL1, XL2), YL is max(YL1, YL2),
  XR is min(XR1, XR2), YU is min(YU1, YU2),
  XL < XR, YL < YU. % not empty
```
(c)

Figure 5. Constraint Handling Rules code for Popular: (a) the `inside` constraint; (b) a recursive procedure for equating base stations; (c) the extended solver, which works with unions of polygons.

precision (see Figure 5c). This corresponds to a disjunctive constraint of the form `inside(S, R1) or inside(S, R2) or... or inside(S, Rn)`, which is more compactly implemented as `inside(S, [R1, R2, ..., Rn])`. In the figure, `Geom` stands for "geometrical object."

We can quickly adapt this solver to work with geometric objects other than rectangles by changing the definition of `intersect_geom/3`. Also, the lifting to three dimensions amounted to just adding a third coordinate and code analogous to the code for the first two dimensions. The solver's simplicity does not mean primitiveness or triviality. Rather, it illustrates CHR's power, because implementing the functionality in a hard-wired black-box solver would be quite difficult.

However, the initial nondisjunctive case already has problems: Finite domains are in principle applicable; however, coordinates would have to be rounded to integers. Also, we found that for our application, Eclipse's built-in finite-domain solver was slightly slower than the CHR implementation. Using linear polynomial constraints would be overkill and thus inefficient. Interval arithmetic[4] can express the required constraints more adequately—even when we move from rectangles to geometric objects that are described by nonlinear equations. However, the disjunctive geometric constraints that we used would require recasting using auxiliary variables, which is expensive, is error-prone, and limits the amount of propagation.

Performance

For a typical office building, Popular finds an optimal placement within a few minutes. This is impressive because we implemented everything (including ray tracing and a graphical user interface) in a constraint-logic programming language. The CLP code is approximately 4,000 lines, with more than half of it for graphics and the user interface. The overall quality of the placements produced is comparable to those produced by a human expert. Placement precision is within 0.7 meters. It is influenced by the underlying path-loss model with its fading reserve, by the number of rays used in the simulation, and by the approximation of radio cells by unions of rectangles.

While the simulation phase has linear complexity in the number of test points, the optimization phase has theoretically exponential complexity. Our practical experience shows, however, that the actual complexity is much lower. As we mentioned earlier, we speed up ray tracing by limiting the number of rays to 128 and using binary search for each ray's threshold. The labeling benefits from heuristics that take into account the problem's geometric nature. The average runtime on a Sun SparcStation 10 was almost linear in the number of walls and test points, with about 25 milliseconds per wall or test point. In a big building, this number might reach several thousand milliseconds, resulting in computation times of about a minute for placement of up to 25 base stations.

The CHR library allowed rapid, flexible, and efficient implementation of the necessary constraints. It let us extend the application from rectangles to unions of rectangles and from 2D to 3D. Also, restricting the base station locations to walls or near ceilings or to aisles for ease of installation and maintenance was just a matter of constraining the base station positions to the union of the allowed spaces.

POPULAR HAS RAISED SOME COMmercial interest. We intend to apply our approach to plan sender locations for last-mile telecommunications and for outdoor mobile communication where obstacles are numerous—for example, in mountainous terrain. These problems are on a much larger scale than that of the problem presented in this article, so their characteristics are different. However, we are confident that the constraint approach will also be useful here. ■

Acknowledgments

The authors did this work when they were at the European Computer-Industry Research Centre (ECRC), Munich, Germany. Popular was developed in collaboration with industry and research institutions in Germany: the Siemens Research and Development Department, the Siemens Personal Networks Department, the ECRC, and the Institute of Communication Networks at the Aachen University of Technology.

References

1. *Building Penetration Losses*, Report COST 231 TD (90) 116, COST 231 Propagation Models Subgroup, Darmstadt, Germany, 1990.

2. M. Wallace, S. Novello, and J. Schimpf, "ECLiPSe: A Platform for Constraint Logic

Further reading and related work

This article is a revised version of a previous paper[1] and is a companion paper to another paper[2] that does not describe the implementation but concentrates on the model used and its features. Kim Marriott and Peter Stuckey provide a complete introduction to constraint programming[3]; Mark Wallace has written a survey on its applications.[4]

Taking advantage of state-of-the-art techniques for programmable application-oriented constraint solving, Popular was among the first practical tools that could optimally plan wireless communication networks. While we were working on Popular, Steven Fortune and his colleagues were developing the WiSE tool,[5] which has exactly the same functionality. WiSE comprises approximately 7,500 lines of C++. For optimization, it uses an adaptation of the Nelder-Mead direct-search method that optimizes the percentage of the building covered. WiSE is patented; Lucent Technologies has used it commercially since 1997 to plan their Definity Wireless Business System/PWT (Personal Wireless Telecommunications). Dimitrios Stamatelos and Anthony Ephremides have developed another approach that uses the Nelder-Mead method for continuous space and Hopfield neural networks for modeling in discrete space.[6] They also briefly mention a tool called IWNDT, which is programmed in C.

References

1. T. Frühwirth and P. Brisset, "Optimal Planning of Digital Cordless Telecommunication Systems," *Proc. PACT '97: Third Int'l Conf. Practical Application of Constraint Technology*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1997, pp. 165–176.
2. T. Frühwirth, J.-R. Molwitz, and P. Brisset, "Planning Cordless Business Communication Systems," *IEEE Expert*, Vol. 11, No. 1, Feb. 1996, pp. 50–55.
3. K. Marriott and P. J. Stuckey, *Programming with Constraints*, MIT Press, Cambridge, Mass., 1998.
4. M. Wallace, "Practical Applications of Constraint Programming," *Constraints J.*, Vol.1, Nos. 1 & 2, Sept. 1996, pp. 139–168.
5. S.J. Fortune et al., "WISE Design of Indoor Wireless Systems: Practical Computation and Optimization," *IEEE Computational Science & Eng.*, Vol. 2, No. 1, Spring 1995, pp. 58–68.
6. D. Stamatelos and A. Ephremides, "Spectral Efficiency and Optimal Base Placement for Indoor Wireless Networks," *IEEE J. Selected Areas in Comm.*, Vol. 14, No. 4, May 1996, pp. 651–661.

Programming," *ICL Systems J.*, Vol. 12, No. 1, May 1997, pp. 159–200.

3. T. Frühwirth, "Theory and Practice of Constraint Handling Rules," *J. Logic Programming*, Vol. 37, Nos. 1–3, Oct. 1998, pp. 95–138.
4. F. Benhamou, "Interval Constraint Logic Programming," *Constraint Programming: Basics and Trends*, A. Podelski, ed., *Lecture Notes in Computer Science*, No. 910, Mar. 1995, pp. 1–21.

Thom Frühwirth is a research assistant at Ludwig-Maximilians-Universität in Munich, Germany. His research interests focus on constraint-based programming and constraint reasoning. He previously worked at the European Computer-Industry Research Centre in Munich. He received his habilitation from Ludwig-Maximilians-Universität and his MSc and PhD in computer science from the Technical University of Vienna. Contact him at Ludwig-Maximilians-Universität München, Oettingenstrasse 67, D-80538 Munich, Germany; fruehwir@informatik.uni-muenchen.de/~fruehwir; www.pst.informatik.uni-muenchen.de/~fruehwir.

Pascal Brisset is a teacher at the Ecole Nationale de L'Aviation Civile. His research interests focus on constraint programming for global optimization. He is particularly interested in relating constraints reasoning and stochastic methods (that is, genetic algorithms). He received his PhD in computer science from the University of Rennes, France. Contact him at Ecole Nationale de L'Aviation Civile, 7 Av. Edouard Belin, BP 4005, F-31055 Toulouse Cedex, France; pascal.brisset@recherche.enac.fr.