Autonomous Cyber Agents: Rules For Collaboration

Sarosh N. Talukdar
Carnegie Mellon University, Dept. of Electrical and Computer Engineering, Pittsburgh, PA 15213
talukdar@ece.cmu.edu

Abstract

A cyber agent is any program, machine or person engaged in computer-enabled work. Thus, cyber agents can vary considerably in complexity and intelligence. Can they, despite their variety, be organized to collaborate effectively? Both empirical evidence and theory suggest that they can. Moreover, there seem to be simple rules for designing problem-solving organizations in which collaboration among cyber agents is automatic and scale-effective (adding agents tends to improve solution-quality; adding computers tends to improve solution-speed). This paper develops some of these rules.

1. INTRODUCTION

Computer networks make it possible to interconnect and therefore, organize, large numbers of distributed cyber agents, varying in type from simple programs to skilled humans. Our goal is to develop a class of organizations in which such agents can collaborate easily and effectively. More specifically, our goal is to develop methods for routinely solving arbitrary instances of the following design problem:

Given:
- a problem-to-be-solved (PTBS),
- a heterogeneous set of cyber agents,
- a computer network and interfaces by which the agents can exchange information,

Design:
- an organization (architecture and control strategy) that makes the best possible use of the agents, that is, maximizes the chances that the agents will collaborate to quickly find good solutions to the PTBS.

Our approach to this design problem has been "bottom-up." We began, several years ago, with the design of organizations for the simplest of software agents. Since then, we have progressed to a class of more elaborate organizations, called asynchronous teams, which seem able to accommodate most, if not all, software agents, do not restrict the entry of humans (though effective human participation is yet to be demonstrated), and have proven to be extremely effective in solving difficult optimization problems.

Asynchronous teams are synthesized from a set of organizational primitives. These primitives have been drawn from many, quite different sources, such as, insect societies [1], cellular communities [2], genetic algorithms [3], blackboards [4], simulated annealing [5], tabu search [6], and brainstorming [7]. Only some of the primitives appear in every asynchronous team. They are:
- a multi-problem agenda. (Each team addresses several problems concurrently. This family-of-problems includes the PTBS and those of its relatives whose solutions might influence or otherwise help in solving it.)
- a strongly cyclic data flow. (Trial-solutions to members of the family-of-problems are produced by the agents and made to circulate among them. Agents collaborate by modifying one another's trial-solutions.)
- completely autonomous agents. (Every agent decides entirely for itself which solutions it is going to modify and when.)
- a situation-specific mix of constructors and destroyers. (Cyber agents can be divided into two broad categories: constructors, that specialize in producing new trial-solutions, and destroyers, that specialize in erasing old trial-solutions. There is no inherent bias against either. Rather, the skills of an asynchronous team can be placed where they best fit, in either its constructors or destroyers.)

Some rules for synthesizing asynchronous teams are developed in the succeeding material. Specifically, Section-2 develops a framework for describing collaborative efforts among cyber agents and provides details on the organizational primitives from which asynchronous teams are synthesized; Section-3 distills the contents of Section-2 into a set of design rules; and finally, Section-4 argues that distributed control and mixed

The work reported here was supported in part by the National Science Foundation under Grant Number ECS-9615599, and by DARPA through Contract Number ONR Grant Number N00014-96-1-0854. A shorter version of this paper appeared in the Proceedings of the Thirty-First Hawaii International Conference on System Sciences (HICSS-31), Hawaii, January 1998.

initiative problems should be the next application area for asynchronous teams

2. DEFINITIONS, PRIMITIVES AND MODELS

Agents are characterized by their abilities to perceive, affect and be controlled. Therefore, any agent can be visualized as a partially controllable link between two worlds--one that it perceives or senses (its input-space), the other, that it affects (its output-space). Of course, these worlds may overlap.

2.1. Cyber Agents

A cyber agent is an agent whose input- and output-spaces are maintained by computers, so that perception for a cyber agent reduces to "reading," and action reduces to "writing."

Definition 1. A cyber agent consists of four components:

- a set of computer-maintained input-memories from which the agent can read,
- a set of computer-maintained output-memories to which the agent can write,
- an operator that copies objects from the input memories, transforms them, and writes the results to one or more of its output memories,
- a control system consisting of a selector to choose objects for the operator to transform, and a scheduler to decide when the selector and operator are to work.

Thus, all computer programs are cyber agents, as are all humans, at least while they are engaged in computer-enabled work.

Definition 2. A cyber agent is completely autonomous if its control system is completely self-contained, that is, if it accepts no selection or scheduling instructions from any other agent; otherwise it is, at best, partially autonomous.

As such, completely autonomous cyber agents can do what they want when they want. In particular, they can choose to work in parallel all the time, if enough computers are available.

Definition 3. The work-cycle of a cyber agent consists of the following sequence: read (copy) a set of objects from its input-memory, modify these objects, and write the results to its output-memory.

In some cases, the set of objects read by a cyber agent might be empty. But the work that a cyber agent does always causes the population of objects in its output-memory to change.

Definition 4. A cyber agent is a destroyer if its output-memory is contained by its input-memory, and if it erases objects from the population in its output-memory; otherwise, it is a constructor.

All the intelligence of a destructive agent is in its control system. Its operator has only to perform the trivial task of erasing objects.

The purpose of a destroyer is to eliminate the mistakes and potential mistakes of constructors by erasing outputs they shouldn't have produced and inputs they shouldn't be considering.

2.2. Collaboration

Definition 5. Two completely autonomous cyber agents are connected if they can exchange data, that is, if an output memory of one is an input memory of the other.

Definition 6. Two completely autonomous cyber agents collaborate whenever they exchange data, whether the exchange is productive or not.

Any collaborative arrangement among two or more completely autonomous cyber agents can be visualized as a "data flow" (Fig. 1).

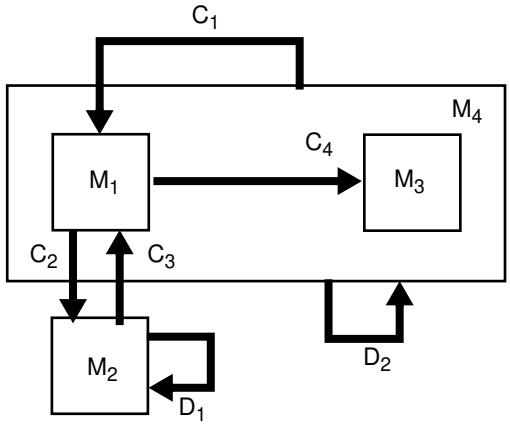

Figure 1. A data flow in which M_1, M_2, M_3, M_4 are memories C_1, C_2, C_3, C_4 are construction agents, and D_1, D_2 are destruction agents. M_1 holds a population of trial solutions to the problem-to-be-solved. M_2 and M_3 hold populations of trial-solutions to related problems. M_4 is the union of M_1 and M_3.

Definition 7. A data flow is a directed hypergraph in which each node is a Venn diagram of overlapping input- and output-memories, and each arc represents the operator and control system of an agent.

Definition 8. A data flow is strongly cyclic if every one of its arcs is in a closed loop.

Definition 9. A data flow is unary if it has one and only

one node representing a single memory that is shared by all the agents in the data flow.

Definition 10. An asynchronous team is a strongly cyclic data flow, that is, a set of completely autonomous cyber agents, connected so their outputs circulate and they interact by modifying one another's outputs.

2.3. Families of Problems

Definition 11 . Two problems are related if a) good solutions to one provide parts of, bounds for, or other clues to good solutions of the second, or b) solutions of one influence solutions of the second. Two or more related problems constitute a family.

Thus, a family of problems can be more than a hierarchical decomposition of the problem-to-be-solved (PTBS). Indeed, a family can include members whose only relationship is that their solutions influence those of the PTBS or provide subtle hints as to their whereabouts. For instance, the problems of designing a car and the process for manufacturing it, form a family, because the solution to the first influences the solution to the second. An optimization problem and the relaxations obtained by dropping some of its constraints, also form a family, because the solutions to the relaxations provide bounds for the solutions of the unrelaxed problem.

There are advantages to simultaneously tackling a family of problems. Progress on the easier members often accelerates progress on the slower, more difficult members, particularly if results are allowed to circulate.

The obvious way to deploy an asynchronous team for concurrently solving a family of problems is to dedicate each memory to one of the problems. This means the memory must be designed to hold a population of trial-solutions to its problem, using a representation that is understood by all the agents that read from or write to that memory.

To start the team, each node is seeded with a population of trial-solutions to its problem. Thereafter, each population evolves as constructors add new members and destroyers erase old members.

2.4. Experience

The key question is: how can good solutions to the family members be made to appear in the evolving populations? After all, there is no coordination among the agents responsible for the evolutions. Rather, each agent, being completely autonomous, does what it wants when it wants, without knowing anything about its colleagues except for the results they produce. Nevertheless, useful asynchronous teams have been developed for a wide variety of optimization and constraint satisfaction problems, including, nonlinear equation solving [8], [9], traveling salesman problems [10], high-rise building design [11], reconfigurable robot design [12], diagnosis of faults in electric networks [13], control of electric networks [14], [15], job-shop-scheduling [16], steel and paper mill scheduling [17], [18], train-scheduling [19], and constraint satisfaction [20]. Not only do these asynchronous teams produce very good solutions, but they appear to be scale-effective.

To define scale-effectiveness and explain why asynchronous teams behave as they do, requires the apparatus that will be developed below.

2.5. Equivalence and Effectiveness

For every memory, M, in a strongly cyclic data flow, there is an equivalent unary data flow. Consider a subgraph, G, that is incident on M, meaning that G consists of an agent A that writes to M, and all the paths that lead from M to A. From the view point of what happens inside M, it makes no difference if this subgraph is replaced by a single super-agent, $\underline{A}$, that both writes to, and reads from M. For instance, the subgraph consisting of C_2, M_2, D_1 and C_3 in Fig 1 can be replaced by a super agent, $\underline{C_3}$, that both reads from and writes to M1. The unary equivalent is obtained when every subgraph that is incident on M has been replaced by a super-agent.The dynamics of M are the same as the dynamics of $\underline{M}$, the only memory in the this equivalent, that is, the population of trial-solutions in M will change with time in the same way as the population in $\underline{M}$. Let:

- S be the space (set) of all possible solutions, good and bad, that can be stored in $\underline{M}$
- q be a measure of solution quality, such that q increases as quality increases
- G_q be the subset of S that contains only solutions of quality q or better
- C, D be the sets of construction and destruction agents that work on $\underline{M}$
- P(0) be a randomly chosen subset of S that is placed in $\underline{M}$ to serve as the initial population of trial-solutions for the agents in C and D to work on
- P(t) be the evolving population of trial-solutions in $\underline{M}$ at time t
- T(q) be the expected value of t for which P(t) and G_q first develop a non-zero intersection, that is, the expected time for P(t) to evolve at least one member of quality q or better.

Definition 12. G_q is reachable if T(q) is finite, that is, if solutions of quality q or better will appear in $\underline{M}$ in an

amount of time whose expected value is finite.

Definition 13. The effectiveness of $\underline{M}$ is the double: $\{q_{max}, T(q_{max})\}$, where q_{max} is the largest value of q such that G_q is reachable.

2.6. Scale-effectiveness

Definition 14. A unary data flow is scale-effective if adding agents increases q_{max}, and adding computers decreases $T(q_{max})$.

Definition 15. A memory, M, is scale-effective if its equivalent unary data flow is scale-effective. An asynchronous team is scale-effective if any of its primary memories is scale-effective. A memory is a primary memory if it is dedicated to the problem-to-be-solved instead of to one of its relatives.

Scale-effectiveness is an extremely desirable property. The problem of improving the performance of a scale-effective organization reduces to one of finding which components (agents, memories and computers) to add. In contrast, a non-scale-effective organization faces the much more difficult problem of finding which of its parts to eliminate or modify before additions can be of benefit. I suspect that most synthetic organizations are not scale-effective: as they grow, there soon comes a point when the addition of another component, no matter how capable it might be, has a negative impact on overall performance. In a scale effective organization there can never be too many agents, memories or computers.

The experience that has been collected in building asynchronous teams indicates that they are often scale-effective. To explain why, the dynamics of the population in any memory will be modeled as a random walk with a distance measure based on the work-cycles of construction agents.

2.7. Construction Space

Construction space, (S, g), is the space of solutions, S, together with an integer-valued function, g, that measures the separation in construction-work-cycles between pairs of subsets of S. The following definition of g is given in terms of the two subsets of principal concern: P(t) and G_q.

Definition 15. $g(P(t), G_q)$, the separation of the set P(t) from the set G_q, is the distance of the closest point in P(t) to G_q, that is, $g(P(t), G_q) = \text{Min } \{f(p, G_q)\}$, $p \in P(t)$, where $f(p, G_q)$, the distance of any point p from the set G_q, is the minimum number of work-cycles by construction agents necessary to improve the quality of p to a level of q or better, that is, to modify p till it becomes a member of G_q.

Separation measured in this way has two noteworthy properties. First, solution-quality and separation are not directly related; the highest quality member of P(t) is not necessarily its closest member to G_q. Second, separation depends on the skills of the operators in C, the set of construction agents. To illustrate, consider two peaks (maxima) in the quality surface, s_1 and s_2. Suppose that the quality of s_1 is only slightly lower than that of s_2. Suppose that C contains only greedy, hill-climbing operators that are unable to descend from a high point in order to reach an even higher point. Then, there is no path from s_1 to s_2. In other words, the distance from s_1 to s_2 is infinite. However, if C is expanded to include operators that can go down-hill, then the distance between s_1 and s_2 will become finite. In general, as agents with new and useful skills are added to C, points of lesser quality draw closer to points of greater quality. Let:

$t_1, t_2, \ldots$	be the discrete points in time at which P(t) changes through the actions of agents. Note: $t_{n+1} - t_n$ is at most the time required for the work-cycle of the slowest agent, because the agents can work in parallel.
$\{P(t_n)\}=$	$P(0), P(t_1), \ldots, P(t_N)$, be a trajectory of successive populations that reaches G_q in N steps
g_n	be the distance of $P(t_n)$ from G_q, that is, $g_n = g(P(t_n), G_q)$
$f(t_n)$	be the probability that g_{n+1} is smaller than g_n
$r(t_n)$	be the probability that g_{n+1} is larger than g_n, that is, the probability that the agents have made a mistake, causing $P(t_{n+1})$ to be further away from G_q than its predecessor, $P(t_n)$
$\lambda(t_n) =$	$f(t_n) - r(t_n)$, be the drift at t_n.

The population's trajectory can be thought of as a random walk. Hence, the expected value of N is finite if g_0 is finite and if $\lambda(t_n)$ is positive; moreover, N decreases as $\lambda(t_n)$ increases. In other words:

- sufficient conditions for obtaining solutions of arbitrarily high quality, q, are: the distance of the initial population from these solutions, g_0, must be finite, and the drift at all points along the developing trajectory, $\lambda(t_n)$, must be positive.
- the time needed to reach these solutions decreases with decreases in g_0, increases in $\lambda(t_n)$, and increases in the number of computers available for the agents to use (at least as long as the addition of computers causes a reduction in the average time required for an agent-work-cycle).

How can g_0 and $\lambda(t_n)$, be improved? An analysis of an idealized version of a unary data flow [21] provides two clues. First, construction and destruction are dual processes; increasing the skills of either C or D (by adding either construction or destruction agents with appropriate

new skills) can be used to improve both g_0 and $\lambda(t_n)$. Second, there are no restrictions on the complexity or phyla of the agents that are added; the additions can mix simple and complex agents, or human and mechanical agents.

2.8. Control Systems for Computer-Based Agents

The active components of a computer-based -cyber agent are its operator and its control system. If the agent is a destroyer, its operator is trivial. If it is a constructor, its operator is likely to be problem-specific. But the control system is likely to have problem-independent features, that is, features which can be designed by general rules. In the search for such rules, we have made some progress in input-selection, but have yet to investigate scheduling, the other part of agent-control.

In selecting the inputs for an operator to modify, experience suggests that a symmetrical adaptation of the solution-acceptance-strategy used in simulated annealing, works well enough to serve as a default strategy, that is, as the strategy for situations where no better strategies are evident.

Definition 17 . Boltzmann selection is random selection from a Boltzmann distribution of solutions with the bias in favor of the higher quality solutions for constructors, and lower quality solutions for destroyers.

For quasi-repetitive problems [22], that is, different instances of the same general problem, such as different instances of nonlinear optimization, [23], [24] describe procedures by which control systems can learn what they are supposed to do from previous work.

3. SUMMARY

3.1. Rules for Collaboration

The preceding material can be distilled into the following rules for designing collaborative, problem - solving organizations for distributed cyber agents.

If the agents are completely autonomous, if they are connected into a strongly cyclic data flow, and if they use Boltzmann or better selection strategies, then it is likely that:

- the data flow will be open; adding agents, memories and computers will be relatively easy.
- the data flow will be scale-effective; adding agents and memories will tend to improve solution-quality, adding computers will tend to improve solution-speed.
- extending the family of problems being concurrently tackled will improve solution-quality for all the problems.
- construction and destruction are duals; adept destruction agents can compensate for inept construction agents, and vice-versa.
- neither scale-effectiveness nor duality are dependent on the granularity or phyla of the agents; therefore, large agents can be mixed with small ones, human agents with mechanical ones.

3.2. The Bases for the Rules

The rules are based, in part, on experience with asynchronous teams [8]-[20] and, in part, on the analysis of unary data flows [21]. They are not absolute. Undoubtedly, there are situations where they will prove to be untrue. Also, their coverage is very general (details being left to the organization designer's intuition) and incomplete (for instance, they offer no recommendations for agent-scheduling, an area that remains to be investigated).

Taken one at a time, the bases for each rule is as follows.

If an agent comes with its own, self-contained control system, then it can be added to an organization without having to instruct it in what is to do, or changing the organization in any other way. Thus, organizations composed of completely autonomous agents should be more open than others. Experience confirms this.

Experience also suggests that the more diverse the skills of the construction agents, the higher the quality of the solutions that will be found. Analysis confirms this by showing that increases in diversity produce contractions in construction space, causing solutions of lesser quality to draw closer to solutions of greater quality.

Increasing numbers of memories can be used to implement beneficial strategies that have been learned for quasi-repetitive tasks [23], and to extend the family of problems being tackled. Experience suggests that the interactions among concurrent efforts to solve members of extended families tend to produce better solutions than are obtained by focusing on a single, difficult member.

The successes of genetic algorithms and simulated annealing suggest that skillful destruction can be useful in finding good solutions. Analysis indicates that in strongly-cyclic data flows, skillful destruction is just as effective as skillful construction and substitutable, one for the other.

Analysis indicates no restrictions on agents by granularity or phyla. Experience confirms that big and small programs can be mixed without penalty or difficulty. But the possibility of combining humans and programs into effective collaborations remains to be investigated.

4. FUTURE APPLICATIONS:

DISTRIBUTED CONTROL AND MIXED INITIATIVE PROBLEMS

The asynchronous teams developed so far have been for off-line optimization problems [8]-[20], such as design, planning and scheduling. Where else might asynchronous teams be of use? On-line optimization, particularly, distributed control and mixed initiative problems (DCMIPs), are possibilities that come to mind.

The essence of a DCMIP is how to make decisions that have global consequences with less than global information. More specifically, a DCMIP can be thought of as a number of coupled subsystems, each with its own controller. The response of each subsystem is influenced by the subsystems to which it is coupled, which in turn, are influenced by the subsystems to which they are coupled, and so on. The problem for the local controller is to ensure that the response of its subsystem is appropriate, even though it cannot sense the states of all the other subsystems, and therefore, cannot determine their influences. The collection of all the local-controller-problems forms a family whose members must be solved concurrently, not unlike the members of families that are now quite routinely solved by asynchronous teams engaged in off-line optimization. This correspondence suggests that asynchronous teams will work as well for DCIMPs as they have for off-line optimizations. Moreover, DCIMPs provide a natural setting for combining programs with humans to test the hypothesis that these two very different types of cyber agents can collaborate effectively and as equals.

REFERENCES

[1] G.F. Oster and E.O. Wilson, "Caste and Ecology in the Social Insects," Princeton University Press, Princeton, NJ, 1978.

[2] A. Kerr, Jr., "Subacute Bacterial Endocardites," Charles C. Thomas, Springfield, IL, 1955.

[3] "Handbook of Genetic Algorithms," edited by L. Davis, Van Nostrand Reinhold, 1991

[4] H. P. Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, Parts I and II, AI Magazine, 7:2 and 7:3, 1986.

[5] S. Kirkpatrick, C.D. Gelatt, and M.P. Cecchi, "Optimization by Simulated Annealing," Science, Vol. 220, Number 4598, May, 1983.

[6] F. Glover, "Tabu Search-Parts I and II," ORSA Journal of Computing, Vol. 1. No. 3, Summer 1989 and Vol. 2, No. 1, Winter 1990.

[7] S. Pugh, "Total Design," Addison Wesley, 1990.

[8] P. S. deSouza and S.N. Talukdar, "Genetic Algorithms in Asynchronous Teams," Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991.

[9] S. N. Talukdar, S. S. Pyo and T. Giras, "Asynchronous Procedures for Parallel Processing," IEEE Trans. on PAS, Vol. PAS-102, NO 11, Nov. 1983.

[10] P. de Souza, "Asynchronous Organizations for Multi-Algorithm Problems," Ph. D. Dissertation, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1993.

[11] R.W. Quadrel, "Asynchronous Design Environments: Architecture and Behavior," Ph. D. Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.

[12] S. Murthy, "Synergy in Cooperating Agents: Designing Manipulators from Task Specifications," Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 1992.

[13] C.L. Chen, "Bayesian Nets and A-Teams for Power System Fault Diagnosis," Ph. D. Dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, 1992.

[14] S. N. Talukdar, V.C. Ramesh, "A Parallel Global Optimization Algorithm and its Application to the CCOPF Problem," Proceedings of the Power Industry Computer Applications Conference, Phoenix, May, 1993.

[15] P. Avila-Abascal and S. N. Talukdar, "Cooperative Algorithms and Abductive Causal Networks for the Automatic Generation of Intelligent Substation Alarm Processors", Proceedings of ISCAS-96.

[16] S. Y. Chen, S. N. Talukdar, N. M. Sadeh, "Job-Shop-Scheduling by a Team of Asynchronous Agents," IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control, Chambery, France, 1993.

[17] J. Rachlin, F. Wu, S. Murthy, S. Talukdar, M. Sturzenbecker, R. Akkiraju, R. Fuhrer, A. Aggarwal, J. Yeh, R. Henry and R. Jayaraman, "Forest View: A System For Integrated Scheduling In Complex Manufacturing Domains," IBM report, 1996.

[18] H. Lee, S. Murthy, W. Haider, D. Morse, "Primary Production Scheduling in Steelmaking Industries," IBM report, 1995.

[19] C. K. Tsen, "Solving Train Scheduling Problems Using A-Teams," Ph.D. dissertation, Electrical and Computer engineering Department, CMU, Pittsburgh, 1995.

[20] S. R. Gorti, S Humair, R. D. Sriram, S. Talukdar, S. Murthy, "Solving Constraint Satisfaction Problems Using A-Teams," to appear in AI-EDAM.

[21] S. N. Talukdar, L. Baerentzen, A. Gove, P. S. deSouza,

"Asynchronous Teams: Cooperation Schemes for Autonomous Agents," to appear in the Journal of Heuristics, and visible at: www.ece.cmu.edu/afs/ece/usr/talukdar/heuristics.ps.

[22] K. M. Carley, "Computational and Mathematical Organization Theory: Perspectives and Directions," Journal of Computational and Mathematical Organizational Theory, Vol. 1, No. 1, Oct. 1995

[23] L. Baerentzen and S.N. Talukdar, "Improving Cooperation Among Autonomous Agents in Asynchronous Teams," submitted to the Journal of Computational and Mathematical Organizational Theory.

[24] L. Baerentzen, P. Avila and S.N. Talukdar, "Learning Network Designs for Asynchronous Teams," in Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer-Verlag