Controlling generate & test in any time

Carl-Helmut Coulon\textsuperscript{\textdegree} and Frank van Harmelen\textsuperscript{\textdegree} and Werner Karbach\textsuperscript{\textdegree} and Angi Voß\textsuperscript{\textdegree}

\textsuperscript{\textdegree}German National Research Institute for Computer-Science (GMD)
P.O. Box 1316, D-5205 Sankt Augustin

\textsuperscript{\textdegree}University of Amsterdam
Roeterstraat 15, NL-1018 WB Amsterdam
e-mail: avoss@gmdzi.gmd.de

Abstract. Most problem solvers have a one-dimensional stop criterion: compute the correct and complete solution. Incremental algorithms can be interrupted at any time, returning a result that is more accurate the more time has been available. They allow the introduction of time as a new dimension into stop criteria. We can now define a system's utility in terms of the quality of its results and the time required to produce them. However, optimising utility introduces a new degree of complexity into our systems. To cope with it, we would like to separate the performance system to be optimised from utility management.

Russell has proposed a completely generic precompilation approach which we show to be unsatisfactory for a generate & test problem solver. Analysing this type of systems we present four different strategies, which require different information and result in different behaviours. The strategy most suitable to our application requires on-line information, and hence had to be implemented by a meta-system rather than a precompiler. We conclude that universal utility managers are limited in power and are often inferior to more specialised though still generic ones\textsuperscript{1}.

1 Motivation

KBS must take time into account: Knowledge-based systems are often built to cope with really hard problems. Although AI is famous for tackling NP-hard problems, its systems usually do not consider the time that any benevolent user may be ready to wait. They are designed and fine-tuned to achieve a fixed stop criterion. For example, most diagnosis systems stop when they reach a leaf in the diagnosis hierarchy. However, if the problem space of such systems is equipped with a notion of incomplete, approximate or partial solutions, the user might sometimes prefer a quick, though approximate, solution.

\textsuperscript{1} The research reported here was carried out in the course of the REFLECT project. This project is partially funded by the Esprit Basic Research Programme of the Commission of the European Communities as project number 3178. The partners in this project are the University of Amsterdam (NL), the German National Research Institute for Computer-Science GMD (D), the Netherlands Energy Research Foundation ECN (NL), and BSR Consulting(D).
Entering a new dimension of utility: Real time systems have recently been defined to be systems whose utility gracefully decreases the shorter they are run [2] [1]. They are based on incremental or interruptible anytime-algorithms whose quality of the output increases over time. These are by no means rare creatures. In principle, every loop is a candidate to be turned into an incremental procedure. Incremental, interruptible, or anytime algorithms introduce a new degree of freedom when defining system utility. Instead of concentrating on the ultimate, most correct solution, we can try to obtain maximal quality within a given time span, or try to reach a minimum quality as quickly as possible, or combine both. Thus, the utility of a system becomes a function of time $U(t)$.

Utility should be handled orthogonally: This new degree of freedom must be used carefully, so as not to introduce a new dimension of complexity into our systems. Ideally, a KBS should be designed as before, and utility optimisation should be handled by a separate component. Beside the utility function to be achieved, such a utility manager needs certain information about the performance of the underlying system. If all relevant information is available before run time, a precompiler would do perfectly.

A utility manager for generate & test: Generate & test is a frequently employed problem solving method in AI: in diagnosis, hypotheses are generated and tested; in search, successor states are generated and evaluated; in design, solutions are proposed & revised; in planning plans are generated and tested by execution. In this paper, we present the principles of generic utility management for the class of generate & test problem solvers. We elaborate four strategies of switching between generation and test in order to maximise the number of solutions given a maximal time limit. The goal of this paper is not to present sophisticated strategies for controlling generate & test. In fact, with the possible exception of the fourth strategy, the strategies we discuss are rather obvious. Instead, the goal of this paper is to present a method of comparing such strategies, and to introduce the parameters that are involved in such comparisons.

A comparison to Russell and colleagues will round off the paper. Although developed independently, their motivation on utility is very similar to our's. In [4] they propose a precompiler separating the algorithmic design of real-time systems from optimising their utility. The latter task is automated based on so-called performance profiles for the basic algorithms. Although we appreciate their intention, we doubt the practicality of their assumptions. While they derive the overall performance profile of a system from those of its basic algorithms, we would like to specify the overall performance profile without having to supply any profiles for the basic steps. Moreover, we have several potential overall performance profiles, but cannot determine the right one statically, so that precompilation is not suitable. In the meantime Russel's & Zilberstein's system maintains several performance profiles and introduces a monitoring component which switches between them at run-time. But we still see the problem of determining which profile to apply in a specific situation.

---

2 Personal communication.