CBack
User Guide

Version 1.0 (August 1995)

by
Keld Helsgaun
E-mail: keld@ruc.dk

1. Introduction

Backtrack programming is a well-known technique for solving combinatorial search problems. The search is organized as a multi-stage decision process where, at each stage, a choice among a number of alternatives is to be made. Whenever it is found that the previous choices cannot possibly lead to a solution, the algorithm *backtracks*, that is to say, re-establishes its state exactly as it was at the most recent choice point and chooses the next untried alternative at this point. If all alternatives have been tried, the algorithm backtracks to the previous choice point.

In addition to the usual depth-first search strategy, CBack provides for the more general heuristic best-first search strategy.

CBack is a simple tool for backtrack programming in the programming language C. The tool is a library consisting of a relatively small collection of program components, all written in the ANSI standard of C. CBack is highly portable and may easily be ported to most computer architectures and C compilers.

The tool is described in detail in the paper

> K. Helsgaun,
> "CBack: A Simple Tool for Backtrack Programming in C",
> Softw. pract. exp., Vol. 25, No. 8, 1995 (pp. 905-934).

This paper may be found in pdf-format in the directory DOC.

2. Installation

The software is available in two formats:

CBack-1.0.tgz	(gzipped tar file, ≈ 100 KB)
CBack-1.0.sit	(Stuffit archive, ≈ 100 KB)

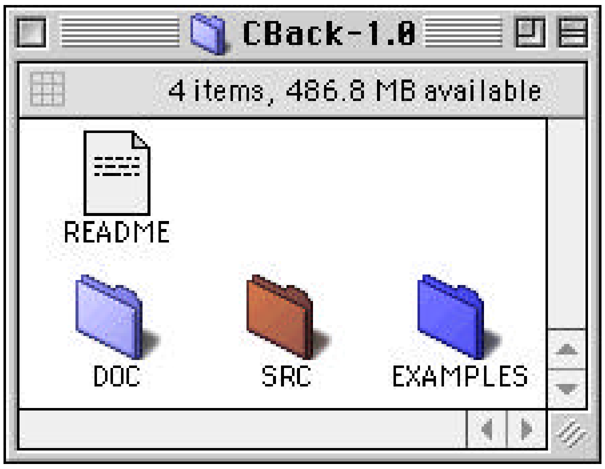
If a UNIX machine is used, download the software in the first format. Next execute the following UNIX commands:

```
gzip -d CBack-1.0.tgz
tar xvf CBack-1.0.tar
```

If a MacOS or a Windows machine is used, download the software in the second format. Next unstuff it with StuffIt Expander™ (freeware available at http://www.aladdinsys.com).

The code is distributed for research use. The author reserves all rights to the code.

On a MacOs machine the following files and folders can be found in the folder CBack-1.0. Similar files and directories can be found on a Windows machine and on a UNIX machine.



The `README` file contains instructions for installing the software.

The `DOC` folder contains the following documentation: (1) `CBack_GUIDE.pdf`, this user guide, and (2) `PAPER.pdf`, a paper that describes the use and implementation of the software.

The `SRC` folder contains source code of CBack. The code, as presented in the paper, is available in the two files `CBack.h` and `CBack.c`. In this code the priority queue of program states is represented a linear linked list. Three alternative implementations, `CBack.skew.c`, `CBack.pheap.c` and `CBack.splay.c`, in which the priority queue is represented by a skew heap, a pheap and a splay tree, respectively, are also provided in the `SRC` folder.

The `EXAMPLES` folder contains the following example programs:

`8Q1.c`, `NQ1.c`, `NQ2.c`, `NQ3.c`, `NQ3s.c` (in the directory `NQUEENS`):

> These programs solves the 8-queens and N-queens problem as described in the `CBack` paper (pp. 907-914).

`Syntax.c`:

> This program performs syntax analysis as described in the paper (pp. 914-915).

`Match.c`:

> The string pattern matching program presented in the paper (pp. 915-916).

`Puzzle15.c` (in the directory `15PUZZLE`):

> This program solves the 15-puzzle problem as described in the paper (pp. 921-924).

`Puzzle15.Korf.c` (in the directory `15PUZZLE`):

> This program solves the 100 instances of the 15-puzzle problems given in the paper:
>
> > R.E. Korf, Depth-First Iterative-Deepening:
> > "An Optimal Admissible Tree Search",
> > *Artificial Intelligence* 27 (1985) 97-109..
>
> On a 233 MHz G3 Macintosh, using `CBack.pheap.c`, the program found solutions to all these problem with an average execution time of 0.15 seconds. The average number of moves was 76.51 (the average of the shortest possible solutions is 53.05).
>
> The 100 problems can be found in the file `KorfProblems`.

`Puzze15.Korf.IDA.c` (in the directory `15PUZZLE`):

> This program find the shortest possible solutions to Korf's problems using iterative deepening search. On a 233 MHz G3 Macintosh the program produced optimal solution with an average execution time of 190 seconds.

`SENDMOREMONEY1.c, SENDMOREMONEY2.c` (in the directory `CRYPTARITHM`):

In a cryptarithm, numbers are represented by replacing their digits by letters; a given letter consistently represents the same digit and different letters represent different digits.

This program solves the following cryptarithm.

```
    S E N D
+   M O R E
  M O N E Y
```

The second version of these programs is the most efficient (by backtracking earlier and making use of the `NextChoice` function).

`CRYPTARITHM.c` (in the directory `CRYPTARITHM`):

This program generalizes the two previous programs. It can solve cryptarithms involving sums (as in the SEND-MORE-MONEY problem). For example, the program can be used to solve the following cryptarithm:

```
        S I X
+       S I X
    S E V E N
  T W E N T Y
```

`NQueen.c`, `NQueen.perm.c` (in the directory `PASCAL`):

These two programs shows how `CBack` can be used with Pascal. They both solve the N-queens problem, but in two different ways.

The Pascal interface is provided in the file `Backtracking.p`.

The software has been written in the ANSI standard for C. If the C compiler concerned does not fulfil this standard, some simple changes are necessary:

- Replace all occurrences of `void*` with `char*`.
- Rewrite all function declarations to the old form (i.e. without prototypes).
- If `<stddef.h>` is not available, then insert the following line in the beginning of `CBack.h`:

  ```
  typedef unsigned long int size_t;
  ```

- If `<stdlib.h>` is not available, then use `<malloc.h>` instead.
- If the function `labs` is not available, then use `abs` instead (in the macro definition of `StackSize` in the beginning of `CBack.c`).

The code may now be compiled and tested with some simple examples (e.g. program 8Q in Figure 2). It will, however, be necessary to use the macro `Backtracking`. In the program the function name `main` is changed, for example to `Problem`, and the following line is appended to the program:

```
main() Backtracking(Problem())
```

If C's runtime stack always has its bottom at the same address, then the `Backtracking` macro can be made superfluous (in most applications). After having executed the following small program

```
main() Backtracking(printf("%lx\n",StackBottom))
```

the address written by the program is inserted as the initial value for the variable `StackBottom` (in the beginning of `CBack.c`).

Some C systems do not always keep the runtime stack up-to-date but keep some of the variable values in registers. This is for example the case for C systems on Sun machines. If this is the case, the macro `Synchronize` must be used; the comment characters are simply removed from the macro definition (in the beginning of `CBack.c`).