A Tail of 2n Cities: Performing Combinatorial Optimisation using Linked Lists on Special Purpose Computers

Abramson, David; Logothetis, Paul; Randall, Marcus; Postula, Adam

Published in:
International Conference on Computational Intelligence and Multimedia Applications 1998 (ICCIMA '98)

DOI:
10.1142/9789814528948

A *TAIL* OF 2^N CITIES: PERFORMING COMBINATORIAL OPTIMISATION USING LINKED LISTS ON SPECIAL PURPOSE COMPUTERS

DAVID ABRAMSON
PAUL LOGOTHETIS
Department of Digital Systems
Monash University
Wellington Rd, Clayton, 3168, Australia
E-mail: davida@dgs.monash.edu.au
http://www.dgs.monash.edu.au/~davida

MARCUS RANDALL
School of Applied Science
Griffith University
Gold Coast, Bundall 9726, Qld, Australia
E-mail: m.randall@eas.gu.edu.au

ADAM POSTULA
Department of Computer and Electrical Engineering
University of Queensland
St Lucia, Qld, Australia
E-mail: adam@elec.uq.edu.au

This paper addresses the development of general purpose meta-heuristic based combinatorial optimisation algorithms. With this system, a user can specify a problem in a high level algebraic language based on linked list data structures, rather than conventional vector notation. The new formulation is much more concise than the vector form, and lends itself to search heuristics based on local neighbourhood operators . The specification is then compiled into C code, and a unique solver is generated for each particular problem. Currently, search engines for simulated annealing, greedy search and Tabu search have been implemented, and good results have been achieved over a wide range of optimisation problems.

We have also implemented a special purpose computer that solves one particular optimisation problem formulated using this technique, namely the travelling salesman problem. Solvers have been produced for simulated annealing, greedy and Tabu search, and a speedup up to 16 times has been achieved over a high-end workstation.

1 Introduction

There are many real world scheduling and planning tasks which can be classified as combinatorial optimisation problems (COPs). These can usually be formulated as a mathematical problem of minimising some cost function subject to a number of constraints. The constraints may specify bounds on the actual problem, but also restrict the solution to ones which are feasible. Usually, such problems are NP-hard, and thus, whilst it is possible to find exact solutions to specific problems, in general only approximate solutions can be found. There are many algorithms which have been proposed for finding approximate solutions to COPs, ranging from special purpose heuristics to general search meta-heuristics such as simulated annealing and Tabu search.

Because most real world COPs have enormous search spaces, it can be quite time consuming to produce solutions of a reasonable quality. Accordingly, researchers have investigated techniques for accelerating algorithms using parallel and high performance computer systems[9]. We have experienced quite large speedups by using special purpose computers[1]

General meta-heuristics like simulated annealing have been applied to a wide range of problems. In most cases, the designer must choose an appropriate data structure and a set of local operators which define a search neighbourhood. The variability in representation techniques, and suitable neighbourhood transition operators, has meant that it is usually necessary to develop new code for each problem. Toolkits like the one developed by Ingber[6] have been applied to assist rapid prototyping of simulated annealing codes, however, these still require the development of new programs for each type of problem. There have been very few attempts to develop a general meta-heuristic solver, with the notable exception being Connolly's General Purpose Simulated Annealing[2].

In this paper we explore the design of a general meta-heuristic based solver for a wide range of COPs, and its implementation on a high performance application specific hardware platform. The main goal of this work is to build an environment in which it is possible to specify a range of COPs using an algebraic formulation, and to produce a tailored solver automatically. This removes the need for the development of specific software, allowing very rapid prototyping. Similar techniques have been available for linear programming based solvers for some years[3]. The new system is based on a novel list based data structure

rather than the more conventional vector notation. The second goal of the work is to develop high performance solvers using parallel and special purpose computer architectures. This has only been partly achieved, however, we have demonstrated the implementation of a machine for solving the Travelling Salesman Problem (TSP) on a FPGA based computing platform. Whilst this machine lacks generality, it does demonstrate that reasonably high performance is possible using FPGA based hardware acceleration techniques. To date, we have developed simulated annealing, greedy search and Tabu search based solvers.

The paper begins with a brief discussion about general COPs, followed by an overview of some meta-heuristic solvers. It then introduces a list based formalism which can be used to represent a wide range of COPs using an algebraic specification technique. The paper then outlines a case study involving the implementation a machine for solving the TSP using the list formulations, and presents some preliminary speedup figures.

2 Combinatorial Optimisation Problems

Many COPS can be considered as minimising a cost function subject to a number of constraints. Consider that x is a data structure which represents a solution to a COP, then the problem becomes:

Minimise $f(x)$
st:
$g_i(x) \oplus b_i \qquad 1 \leq i \leq N$
where:
$f(x)$ and $g_i(x)$ are arbitrary functions of a solution x
b is the right hand side vector
N is the number of constraints
$\oplus$ is one of $\leq$, $<$, $=$, $>$, $\geq$, or $\neq$

In most COPs, x contains a set of integers, which can be used to encode a solution to the problem.

3 Meta-Heuristic Search

Over the years a number of heuristics have been proposed for solving COPs. Recently, a range of meta-heuristics have also been applied, for example, simulated annealing [11] genetic algorithms [5] and Tabu search[4]. In many of these algorithms, the problem solved is an

analogue to another problem. For example, in simulated annealing the system being solved is compared to the task of minimising the energy in an atomic lattice, and an annealing algorithm (slow cooling) is applied to the system. In this section we briefly review each of the algorithms, and provide the interested reader with additional references.

3.1 Simulated Annealing

Simulated annealing is derived from the physics of annealing metals. Simulated annealing seeks to minimise an energy function, which in combinatorial optimisation, is the objective function. At the beginning of the annealing there is a high likelihood of accepting any transition regardless of whether or not it improves the solution. This likelihood decreases as the run proceeds. Each transition consists of choosing a variable at random and giving it another (random) value. This process is performed in accordance with an exponential acceptance function based on a parameter called *temperature*. The temperature is decremented until it is quite small and hence very few uphill moves, in which a worse solution may replace the current solution, are accepted. As simulated annealing can make these uphill moves, settling into a local optima is potentially avoided. The way the temperature is controlled is referred to as the *cooling schedule*. For further information about simulated annealing and its variants, see [2, 7, 8, 11].

3.2 Greedy Search

Greedy search is the simplest of the iterative search techniques. Given an initial feasible solution to a problem, greedy search will examine the entire neighbourhood for a solution with a cost better than the current solution. This process is repeated until an improving move cannot be made. As this process will stop at the first local optima it encounters, it is often referred to as *local optima search*.

3.3 Tabu Search

Tabu search is an enhancement of greedy search. The two algorithms differ if a local optimum is encountered as Tabu search will accept the best non-improving move rather than stopping. Therefore, transitions that do not improve the solution quality can be accepted

allowing the algorithm to escape local optima. In order to avoid revisiting previous solutions, a form of cycle avoidance is implemented by a memory structure known as the *Tabu list*. For further information about Tabu search and its variants, see [4].

4 List based Meta-heuristics

Traditionally, there are two main ways of representing a COP. First, an integer vector notation can be used to encode the solution. This approach has been most commonly applied when OR algorithms like linear programming and branch and bound are used. Second, the solution can be encoded in a dynamic data structure, such as a set, graph or list. This approach has been used when specially coded programs are used to find solutions, such as tailored heuristics. The first lends itself to standard packaged solutions because a common structure is used for all problems. However, the dynamic data structure approach usually requires the development of a separate algorithm for each problem.

As part of this work, we have defined a representation technique based on dynamic lists. The technique can best be illustrated by an example, as shown in Figure 1. In this problem, the Generalised Assignment Problem (GAP), jobs are assigned to agents, such that the cost of assignment is minimised, and certain agent capacity limits are enforced. Solutions can be represented using a double nested list of agents, each of which contains a list of associated jobs. Thus, in the example, jobs 1, 3, 5 and 2 are assigned to agent 1.

In the GAP, the objective function is to minimise the total cost of assigning the jobs to the agents and can be expressed as:

$$\sum_{i=1}^{M} \sum_{j=1}^{|x(i)|} C(x(i, j), i) \qquad (1)$$

Where:

x(i,j) is the *j*'th job performed by agent *i*
C(i,j) is the cost of assigning job *i* to agent *j*
M is the number of agents

Whilst this list notation appears similar to conventional vector form used in standard LPs, each sub-list contains a variable number of elements. Thus, the second summation sign in (1) requires a bound

which varies depending on the length of each sub-list (i.e. $|x(i)|$) and changes according to the current solution state. Similarly the constraints concerning the capacity of each agent are formed across list space.

$$\sum_{j=1}^{|x(i)|} a(x(i,j),i) \le b(i) \qquad \forall i \qquad 1 \le i \le M \tag{2}$$

$$|x| = M \tag{3}$$

$$1 \le x(i,j) \le N \qquad \forall i \qquad 1 \le i \le M \qquad \forall j \qquad 1 \le j \le |x(i)| \tag{4}$$

$$count(x) = 1 \tag{5}$$

Where:

$a(i,j)$ is the resource required by agent j to perform job i
$b(i)$ is the capacity of agent i
N is the number of jobs

Equation (3) indicates that there are M sub-lists (one for each agent) while (4) restricts the job numbers to be in the range from 1 to N. Equation (5) ensures that each job is represented exactly once in the solution.

This technique has been applied to a very wide range of problems, and is powerful enough to represent many different COPs.

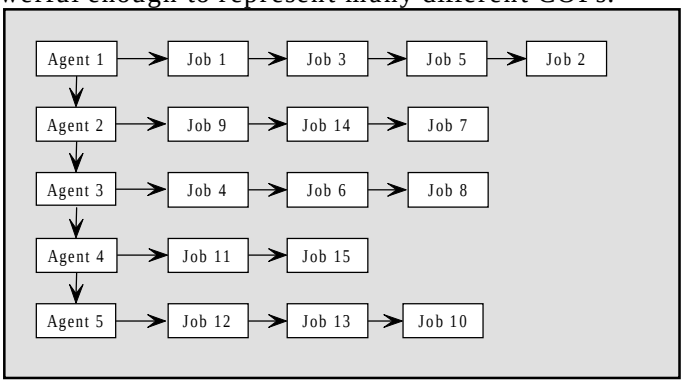

Figure 1: A linked list representation of a GAP

We have produced a system which accepts COPs formulated using a list notation, and which can apply either simulated annealing, greedy search or Tabu search. The system processes the formulation and automatically generates C code, which is then compiled and executed on a conventional computer. In the next section we briefly describe a special purpose computer architecture for the TSP.

5 A Special Purpose Computer for the Traveling Salesman

5.1 The Travelling Salesman Problem

The Travelling Salesman Problem consists of finding the shortest path that a salesman must travel whilst visiting a set of cites only once. Over the years, there have been many techniques developed for solving the problem, and even quite large problems have been solved exactly. It also serves as an excellent test case for our algorithms, because it is quite simple to formulate, and yet is still difficult to solve reasonably large problems to optimality.

The list formulation of the problem is as follows:

$$\text{Minimise } \sum_{i=1}^{N-1} d(x(i), x(i+1)) + d(x(0), x(N-1)) \qquad (6)$$

$$\text{st } x(i) \neq x(j) \qquad \forall\ i,j,\ 0 \leq i,j \leq N\text{-}1,\ i \neq j$$

where:

x contains a list of cities identifiers

d(a,b) returns the distance between cities *a* and *b*.

N is the number of cities

Given this formulation of the TSP, there are 2 main transition operators of interest, namely *swap* and sub-list *reversal*. In the *swap* operator, two cities are interchanged, whereas in the *reversal* operator, the order of the sub-tour between two cities is reversed. Apart from different update algorithms, these two operators also have different expressions for computing the change in cost.

5.2 A Special Purpose Architecture

There is insufficient space in this paper to outline the exact architecture of the TSP machine. Basically, the machine performs a number of iterations in which various local optimisations of the tour are generated, and the change in cost is evaluated. For the TSP it is possible to generate an incremental cost expression, which means the change in cost can be calculated without re-computing the cost of the entire tour. This optimisation means that it is possible to evaluate the change in cost quite quickly. By overlapping various computations, the hardware is

able to compute the change in cost in one machine cycle.

The machine consists of hardware for generating the local optimisations, in this case, either which arcs or cities in the tour are to be exchanged. In simulated annealing, this consists of a random pair of city indexes. In greedy and Tabu search, it consists of all possible pairs of cities. The machine computes the change in cost by using a number of memory structures and arithmetic units. These evaluate an arithmetic expression which gives the change in cost resulting from the change to the tour the tour. In simulated annealing, the interchange is accepted depending on the value returned by the Boltzmann equation, which gives a probability of acceptance at particular temperatures. This equation is evaluated by using special lookup tables loaded when the temperature value is altered. In the greedy search algorithm, all pair wise interchanges are evaluated, and the best is chosen. In Tabu search, all pair wise evaluations are evaluated, and the best one which is not in a Tabu state is chosen. The state of any given move is stored in a special memory called the Tabu memory.

5.3 Performance

We have evaluated the performance of a number of different machines against specially coded C programs. Table 1 summarises the performance of the hardware in which the time per iteration is recorded together with the speedup.

The results show that the performance of the hardware varies widely across the various search algorithms and local exchange operators. The best performance was achieved for simulated annealing, where the hardware is about 16 times faster than the C code. The worst performance was for the Tabu algorithm, in which the hardware is only 2.5 times faster. This poor performance occurs because the small problem size seems to allow the Ultra to store the important data structures totally in the cache, and thus the processor seems to achieve a high throughput. Also, in simulated annealing, the algorithm has to compute the exponential function and generate random numbers, and these functions are optimised and parallelised in hardware. In hardware, the swap operator generally gives higher relative performance than the reverse because the change in cost function has more concurrency.

The performance of Tabu search in hardware could be improved further by implementing a parallel search of the neighbourhood. This

should yield a speedup linear in the level of parallelism, because each neighbourhood transition is independent of the others. In fact, we have implemented a software version of this on a parallel supercomputer, and the performance scales well with the number of processors. We will investigate this technique for improving the performance of the hardware Tabu search code in the future. Further, we would expect to achieve superior performance using more custom circuits and boards than the general purpose FPGA platform used here. For example, the delay of the programmable interconnection switches on the Aptix board is a significant part of the overall computation cycle.

Algorithm	**C code (Sun Ultra 1)**	**Hardware**	**Ratio**
Simulated Annealing - Swap	4.96 μs	300 ns	16.4
Greedy Search - Swap	438 ns	100 ns	4.4
Tabu – Swap	460 ns	100 ns	4.6
Simulated Annealing – Rev	4.72 μs	300 ns	15.7
Greedy Search – Rev	239 ns	100 ns	2.4
Tabu - Rev	248 ns	100 ns	2.5

Table 1: Hardware Performance

Conclusions

In this paper we have discussed a new modeling technique for COPs and have discussed its use with meta-heuristics like simulated annealing, greedy search and Tabu search. The system has proven to be extremely powerful for representing a range of problems. Currently, a software tool is available for translating an arbitrary model into executable C code.

As a related effort, we have produced a hardware implementation of one particular COP, namely the Travelling Salesman Problem. This machine has been implemented for a range of search heuristics. The performance of the hardware varies significantly from about 16 to 2.5 times over an engineering workstation. The performance gain of 16 times would allow a run which previously took 1 hour to be executed in about 4 minutes. The speedup achieved by the Tabu search could be further improved by implementing the hardware as an ASIC rather than using FPGA devices, and also by increasing the amount of internal parallelism.

A major disadvantage of the current system is that it requires the development of new hardware for each problem type. Currently, we are

also investigating ways of automatically generating hardware for a given problem, in the same way that the software is compiled from the algebraic specification.

Acknowledgments

This project is supported by the Australian Research Council as part of a Large Grant. The authors would like to thank Mr David Duke for his contribution to this work, in particular, the implementation of the greedy and Tabu meta-heuristic engines.

7 References

1. Abramson, D.A., "A Very High Speed Architecture to Support Simulated Annealing", IEEE Computer, May 1992, pp 27 - 34.
2. Connolly D.T. (1992) “General purpose simulated annealing”, *Journal of the Operational Research Society*, **43**, 495-505
3. “GAMS home page” at http://www.gams.com, GAMS Development Corporation
4. Glover, F. (1992) “Modern Heuristic Techniques for Combinatorial Problems”, *Blackwell Publishing*
5. Goldberg D.E. (1989) “Genetic Algorithms is search, optimization and machine learning” , Addison Wesely, Reading, 412 pages
6. Ingber L. (1993) “Simulated Annealing:Practice versus Theory”, Computer Modelling, **18**, pp 29-57
7. Johnson D.S., Aragon, L.A., McGeogh and Scheveon C. (1991) “Optimisation by Simulated Annealing: An experimental Evaluation Pt II, Graph Colouring and Number Partitioning”, Operations Research, **39**, pp 378-406
8. Kirkpatrick S., Gelatt D., Vecchi M., (1983): “Optimization by Simulated Annealing”. Science **220**, pp 671-680
9. Mavridou T., Pardalos P.M., Pitsoulis L., Resende M.G.C., (1995) “Parallel Search for Combinatorial Optimization: Genetic Alorithms, Simulated Annealing, Tabu Seach and GRASP”, *Proceedings of Parallel Algorithms for Irregularly Structured Problems*.
10. Osman, I.H., Laporte G. (1996) “Metaheuristics: A Bibliography”, Annals of Operational Research, **63**, pp 513-618
11. van Laarhoven, P.,Aarts E., (1987): “Simulated Annealing: Theory and Applications” D Reidel Publishing Company, Dordecht, 186 pages.