

A General Parallel Tabu Search Algorithm For Combinatorial Optimisation P ^{*}

Marcus Randall¹ and David Abramson²

¹ School of Information Technology, Bond University, QLD 4299, Australia
mrandall@bond.edu.au

² School of Computer Science and Software Engineering, Monash University, VIC
3168, Australia davida@csse.monash.edu.au

Abstract. Tabu Search (TS) is a meta-heuristic search algorithm that is easy to parallelise. Efficient parallelisation of TS can represent a significant saving in the real-time required to solve a problem over an equivalent sequential algorithm. In this study, a general parallel TS algorithm for solving combinatorial optimisation problems (COPs) is presented. The unique feature of our approach is that the TS solves a wide range of COPs expressed in a high level syntax. The benefit of this general code is that it can be used in real-time applications due to its parallel scalability and the fact that it can accept changing problem definitions. After reviewing a number of suitable parallelisation strategies, results are presented that show that good parallel speedup is achieved while efficient solutions to hard COPs are obtained.

Keywords: Combinatorial Optimisation, Parallelisation, Tabu Search.

1 Introduction

This study concentrates on the meta-heuristic known as Tabu Search (TS). TS is a widely used meta-heuristic for solving Combinatorial Optimisation Problems (COPs). An important feature of TS is that it is easy to parallelise, using a variety of different parallelisation schemes. This is unlike the other popular meta-heuristic, Simulated Annealing (SA) which is primarily a sequential algorithm (though some attempts at parallelisation have been made, see [19, 27]).

In this study, an implementation of a general parallel TS engine that uses a unique system based on linked list modeling for describing COPs is outlined. This system is unique as other parallel TS implementations are tailored to specific problems. Linked List modelling is briefly explained in Section 2, but the interested reader is referred to [22, 23] for a more complete description. Section 3 gives the general structure of a tabu search algorithm while 4 reviews a number

^{*} We would like to thank the Queensland Parallel Supercomputing Facility for the use of the IBM SP2 computer. This work was performed under an Australian Research Council Large Grant.

of ways in which the algorithm can be parallelised. After describing our particular implementation of TS (Section 5), it is shown that the solver obtains good solutions to hard COPs with good parallel speedup and efficiency (Section 6 and 7).

2 An Overview of Linked List Modeling of COPs

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 Operations Research 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 vector approach 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.

This work is motivated by the need to solve real-time applications in which problem definitions may change periodically. The aim is to provide a high performance TS platform that can process general problem definitions. In order to achieve this, a representation technique based on dynamic linked lists is used. The technique is discussed in [22, 23]. The resulting general search engine showed that it could achieve the same or better results as tailored heuristics.

Link list modelling can best be illustrated by an example, as shown in Fig. 2. In this problem, the Generalised Assignment Problem (GAP) [10, 21], 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. The nested lists are referred to as *sub-lists*, and each piece of data contained on the list as an *element*. The placement of elements on particular sub-lists or at certain positions on the sub-lists defines the current solution to the problem. To denote that element e is assigned to the i^{th} sub-list at the j^{th} position, the notation $x(i, j) = e$ is used. In this study, we concentrate on problems that can be expressed with one decision list (denoted by x). All other literals in the problem models are considered as constants, vectors and matrices as appropriate.

In the GAP, the objective function is to minimise the total cost of assigning the jobs to the agents and can be expressed as 1

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

Where:

x is the solution list. $x(i, j)$ is the j^{th} job performed by agent i .

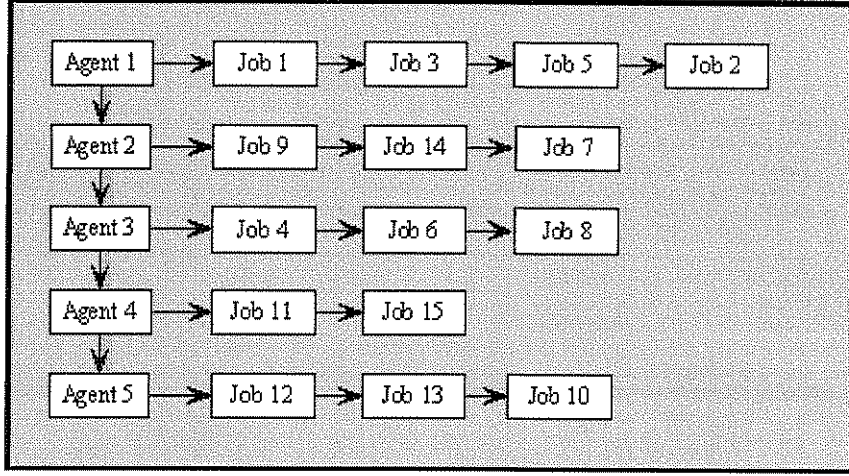


Fig. 1. A linked list representation of a GAP.

C is the cost matrix. $C(i, j)$ 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.

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

$$|x| = M \quad (3)$$

$$1 \leq x(i, j) \leq N \quad \forall i, j \quad 1 \leq i \leq M \quad 1 \leq j \leq |x(i)| \quad (4)$$

$$\min_count(x) = 1 \quad (5)$$

$$\max_count(x) = 1 \quad (6)$$

Where:

a is the resource matrix. $a(i, j)$ is the resource required by agent j to perform job i .

b is the capacity vector. $b(i)$ 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 . $\text{min_count}(x)$ and $\text{max_count}(x)$ return the number of occurrences of the least frequent and most frequent element value on list x respectively. Therefore, equations 5 and 6 ensure that each job is represented exactly once in the solution (i.e. so that two agents do not perform the same job).

This list modelling technique has been applied to a very wide range of problems [22, 23] and is powerful enough to represent many different COPs (some of which are shown in Fig. 2). In order to specify a list model, we have developed a text-based language that allows the model developer to write compact problem descriptions. A problem description consists of an objective function, constraints, list structure and problem data. The language is very similar to GAMS (General Algebraic Modelling System) [8] and is discussed in [22, 23].

Traveling Salesman Problem	Quadratic Assignment Problem	Graph Partitioning Problem	Car Sequen
Bin Packing Problem	Maximum Clique Problem	Knapsack / Multiple Knapsack	Set Partiti
Generalised Assignment Problem	Aircraft landing Problem	Vehicle Routing Problem	Linear Orc
Time Tabling Problems	Hub allocation Problems	Problem Allocation Problem	Single layo
Personal Time Scheduling Problem	N Queens Problem	Number Partitioning	Cutting St

Fig. 2. Some of the problems that have been formulated using list modelling.

3 Tabu Search Overview

TS is a relatively new local search method that has been successfully applied to COPs [14–16]. A local search (or neighbourhood method of search) iteratively makes small changes (referred to as *transitions*) to the solution so that the cost improves over time. [14] describes tabu search as “Tabu search guides ... a heuristic to continue exploration without being confounded by an absence of improving moves, and without falling back into a local optimum from which it previously emerged.”. Thus TS can be thought of as an enhanced and more general version of the well-known hill climbing heuristics (in which only improving transitions are accepted). The unique characteristics of TS are:

Tabu Search can escape local optimum traps: Local optima serve as attractors to search techniques. This is inevitable, as search techniques seek the global optimum. Often search techniques become trapped in one either permanently like hill climbing or a potentially a large number of transitions like simulated annealing. TS overcomes this inherent problem by evaluating the neighbourhood of the current solution, $N(x)$ and choosing the best transition from the ones currently available in $N(x)$ regardless of whether or not

it improves the current solution cost¹. If the transition is non-improving, then the search process has encountered a local optimum and thus begins the process of escape immediately.

Tabu Search can effectively sample the search space: After a transition is made in which a non-improving move is accepted, the new neighbourhood, $N'(x)$, contains the previous state which is now an improving transition. If this is accepted as the next transition, the search can be said to be cycling. TS has a mechanism that overcomes this problem that is referred to as the tabu list. The tabu list stores information that the search can use in order to avoid previously traversed search routes. A transition is considered tabu if it has been recorded on the tabu list and its tabu tenure has not passed. The tabu tenure is the number of iterations that an item on the list stays tabu. As only limited information is recorded on the tabu list, it is possible that the search process will regard solutions that have not been previously encountered as being tabu. In order to counteract this potentially negative effect of the tabu list, TS makes use of one or more aspiration functions. The simplest and most widely used of these functions is a rule that states that a tabu transition is accepted if it produces a superior quality solution compared to those previously encountered. Glover and Laguna [16] discuss a wide variety of aspiration functions.

3.1 Search Over Lists with Tabu Search

As a result of using a list representation, it is possible to apply well-known local search operators to the list structure that have successfully been used in other heuristic codes, for example [18, 20, 21, 25, 26]. There are a variety of ways that a list can be altered in order to form a new solution. However, seven different transition operators have been identified as sufficient to navigate the search space of the list-based formulations. Given the wide variety of standard COPs that have converted into this notation (see Fig. 2), this set appears to be sufficient to find good quality solutions efficiently. The operators (and a description of their TS neighbourhoods) are:

- *Move*: The neighbourhood consists of moving each element from its sub-list to the end of each other sub-list in the solution. For instance, if an element at $x(1,1)$ is in a solution having three sub-lists, two moves involving this element would be to place it at the end of sub-list 2 and sub-list 3.
- *Swap*: The swap neighbourhood is composed of a set of pairs of elements. A pair consists of two different elements of the list. This set contains all possible combinations of elements. Given n elements in a list, this equates to $\frac{(n-1)^2}{2}$ neighbours.

¹ A subset of $N(x)$, $N_s(x)$ ($N_s(x) \subseteq N(x)$) can be used to represent $N(x)$. This is done if the problem is large and subsequently $N(x)$ has many elements [14, 16, 17]. Candidate list strategies are used to calculate $N_s(x)$ [16].

- *Inversion*: The inversion neighbourhood consists of pairs of elements between which the sequence can be reversed. Therefore, both elements of a pair must be located on the same sub-list. The neighbourhood is a subset of the swap neighbourhood.
- *Reposition*: The reposition neighbourhood consists of each combination of element and position on that element’s sub-list. For instance, if sub-list 1 has a length of 4, the possible transitions available to the element at $x(1,1)$ are to reposition it at positions 2,3 and 4 on the sub-list.
- *Add*: The neighbourhood consists of adding each element value in the legal range to the end of each sub-list. For instance, if sub-list 1 contains two elements and the value range is given by $1 \leq x \leq 4$, element $x(1,3)$ can take on the values 1,2,3,4.
- *Drop*: The neighbourhood consists of dropping each element from its place in the current solution. Therefore the size of the neighbourhood is the number of elements in the list structure.
- *Change*: The neighbourhood consists of the set of transitions in which each element in the solution structure changes its value to a different value in the legal range. For instance, if the value range of the solution is $1 \leq x \leq 4$ and $x(1,1) = 2$, the values that $x(1,1)$ can change to are 1,3,4.

Depending on the problem being solved, several of the operators may be appropriate. The solver subsequently allows multiple transition operators to be applied to a particular problem. At each iteration of the tabu search, a transition operator is selected probabilistically. For more information regarding this topic and a description of each of the transition operators, see [22, 23].

3.2 Incremental Cost Calculations

The cost function of many COPs is costly to compute, especially for realistic size problems. The linked list structure makes it possible to compute an incremental cost based on the elements that have changed in the list. This can result in substantial computational savings, particularly for TS in which a large number of neighbours are typically evaluated at each iteration.

Algebraic incremental cost expressions can be developed for problems modeled in the linked list notation. As each of the local search operators alters the solution in a different way, a number of incremental cost expressions may be required for the one problem type. In this general-purpose implementation, incremental cost expressions are automatically derived. However, if incremental costs cannot be established or do not represent substantial savings, they may be omitted. In this event, only the full objective cost function is evaluated. Details of this process are outlined in [22, 23].

4 A Review of Parallelisation Strategies for Tabu Search

In recent years, parallel processing hardware platforms have become increasingly more available. This has meant potential increases in the performance of search

algorithms for COPs. TS is particularly well suited for parallel implementation and the performance is generally scalable. Therefore by using a number of processors, the real (wall clock) time can be substantially reduced over an equivalent sequential program. There are a number of strategies available and these are described in [11, 16]. Combining parallelisation strategies has also proven effective, as demonstrated by [5]. Some of the more common may be summarised as:

1. *Parallel Evaluation of Neighbours*

This method is an implementation of the master slave model and works on the premise that the most computationally expensive part of tabu search is the evaluation of the neighbours (corresponding to low level parallelism). Given P slave processors, the neighbourhood is divided equally among the processors. Each processor performs the transitions along with the evaluation of the cost function and constraints and sends its best neighbour back to the master processor. The master processor then determines the best neighbour among all that it has received (according to the tabu rules) and broadcasts the neighbour so that each slave may update its local copy of the solution. This approach can require considerable communication time because of the master slave configuration and iterative nature of the algorithm [16], but can be easily applied to a range of COPs. This method can be illustrated by considering the Quadratic Assignment Problem (QAP) [9]. Given that a possible transition operator consists of swapping pairs of facilities, each processor can be assigned a subset of these pairs to evaluate. See [26, 13] as examples of the implementation of this method.

2. *Parallel Independent Tabu Searches*

In this approach, a number of sequential tabu searches are run simultaneously across processors on a particular problem. Each search is different as key parameters such as random seed, initial solution or tabu list size are varied. This method is particularly suitable to parallel architectures in which each node behaves as an independent system, such as MIMD (Multiple Instructions Multiple Data). Because of the independence of the searches, no communication is required between the processors. See [25] for an example of the implementation of this method.

3. *Parallel Interacting Tabu Searches*

This approach is similar to method 2, except that at given intervals in the search process, an interaction between the searches occurs [12]. This consists of determining which search has been the most successful and transferring its solution to the other search processes. Each search then continues with an empty tabu list. This approach can have quite a large communication overhead due the necessity of broadcasting entire solution structures.

4. *Search Space Division*

Each processor is assigned a subsection of the search space. A tabu search subsequently explores its subsection and sends back the partial solution to the master process once it has finished [12]. These partial solutions are combined into a final solution. While this method has low communication costs, the process of dividing the search space is very problem specific and may not be possible for all problems. An example of

this deals with the Vehicle Routing Problem [26]. Each processor is assigned a unique vehicle tour to optimise. At the end of the search, each processor sends its tour to the master processor that assembles these tours and computes the overall cost.

5 The Parallel Strategy

The parallel hardware platform used in this study consists of a MIMD IBM SP2 with 22 RS6000 processors. As the list representation for problems is general in nature, methods 1 and 2 outlined in the previous section can be implemented. Because each processor is an independent system, method 2 can be easily achieved by running a number of sequential tabu searches on different nodes of the same computer. Despite the communication cost overhead associated with method 1, this approach has been adopted because its applicability across problem type is required for the general list modelling system and the architecture of the SP2 system. Our parallel code is implemented using the MPI (Message Passing Interface) library version 1.0.

Section 5.1 describes how the transition operator neighbourhoods are divided amongst the processors, while 5.2 illustrates the overall parallel algorithm that has been adopted.

5.1 Division of the Neighbourhood

At each iteration of the tabu search, the master processor computes an appropriate sub-neighbourhood to delegate to each of the slaves based on the transition to perform to the current solution and the number of available processors. Only transitions that result in feasible solutions are considered by the TS implementation proposed here. To ensure an even loading amongst the processors, each processor is assigned $\frac{N}{P}$ neighbours. Here N denotes the number of feasible neighbour transitions while P is the total number of processors.

In the event that $\frac{N}{P}$ is non-integral, the last processor is assigned a different size neighbourhood subset to evaluate than the other processors. This is calculated by the following rule:

if $((\frac{N}{P}P - \text{trunc}(\frac{N}{P}P)) \geq 0.5)$

$S = \text{trunc}(\frac{N}{P}P)$

else

$S = \text{trunc}(\frac{N}{P}P) + 1$

$S_P = N - (P \cdot S)$

Where:

$\text{trunc}(x)$ returns the integer component of x .

S is the number of neighbours the processors 1 through $P - 1$ receives.

S_P is the number of neighbours that processor P receives.

This procedure can lead to a slightly uneven distribution of neighbours to processors. However, as the numerator in $\frac{N}{P}$ becomes large (i.e. more neighbours), the ratio of left over neighbours to allocated neighbours becomes small. Hence, for large size problems, this method of allocation should make little difference to the performance of the TS engine.

subsectionAlgorithm Figure 5.1 shows the inter-processor communication needed for the *parallel evaluation of neighbours* method. The master and slave processes are synchronised by the communication messages. The master's task is to coordinate the entire tabu search process as well as to control key data structures such as the tabu list and solution memory. It is responsible for delegating the neighbourhood evaluation tasks to the slaves as well as determining which neighbour replaces the current solution. In addition, the master also acts as a slave as it evaluates a subsection of the neighbourhood. In order to minimise the amount of communication, each slave keeps a copy of the solution in local memory which is modified after the master broadcasts the solution update. This is more efficient than broadcasting the entire solution to the slaves at every iteration. The master's and slaves' activities are described fully in Figs. 5.1 and 5.1 respectively.

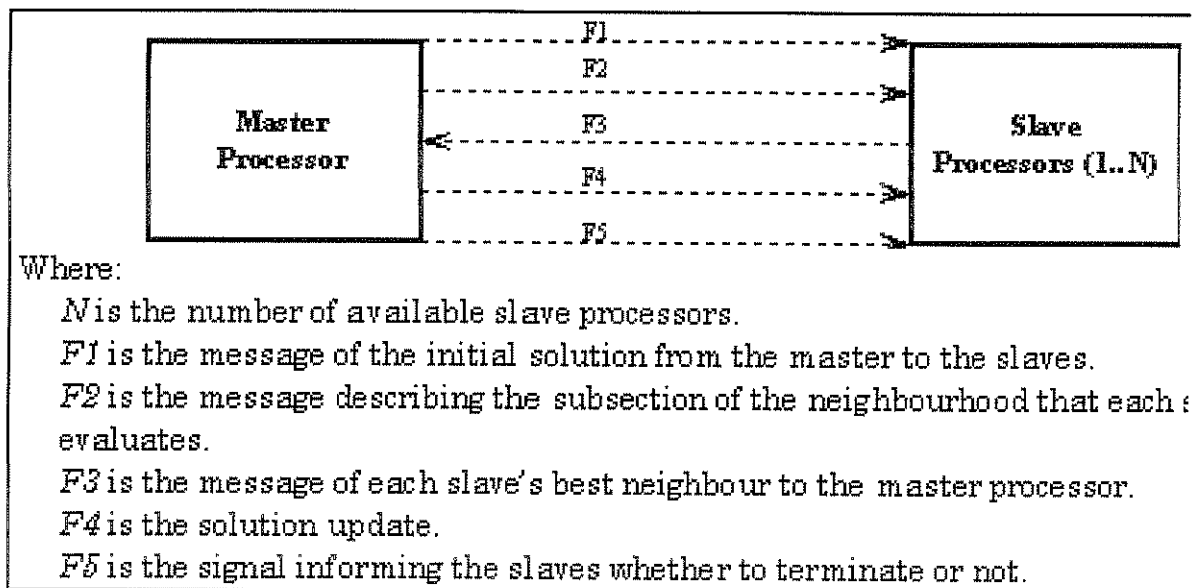


Fig. 3. Block diagram of the communications between the master processor and the slave processors for the parallel TS algorithm.

```

Generate an initial feasible solution;
Broadcast the initial solution to the slaves (send message F1);
While (termination condition not met)
    Partition the neighbourhood into equal sizes and send partition
    details to the slaves (send message F2);
    Evaluate each neighbour from own neighbourhood and retain the
    best neighbour;
    Collect each slave's best neighbour (receive messages F3);
    Determine the most suitable neighbour (using the tabu list
    and aspiration rules) and use it to form the next solution;
    Broadcast the attributes of the chosen transition to the slaves
    (send message F4);
    Update the tabu list;
    Determine if the termination condition has been met;
    Broadcast the termination information to the slaves (send message F5);
End While;
Report best obtained solution;
End.

```

Fig. 4. Pseudocode for the master processor.

```

Receive Initial solution (receive message F1);
While (terminate signal is not "stop")
    Receive details of neighbourhood partition to evaluate
    (receive message F2);
    Evaluate the neighbours in this partition;
    Send the transition attributes of the best neighbour to the
    master processor (send message F3);
    Receive the attributes of the chosen transition and update
    the local copy of the solution (receive message F4);
    Receive the termination information signal (receive message F5);
End While;
End.

```

Fig. 5. Pseudocode for the slave processors.

6 Problems and Methodology

Table 1 lists the problems and problem instances that are used in this study (their list model descriptions appear in [22]). These problems are representative of a broad range of COPs. Each run is allowed at most 2 hours of wall clock time on dedicated nodes in order to find the optimal or best known solution cost. The number of processors is also varied in order to determine the effectiveness of the parallel code. Processor groups vary between 1 and 12 nodes.

Table 1. Problem classes and instances that are used in this study.

Problem	Reference	Instance	Description	Optimal Cost	Best Known Cost
Graph Partitioning Problem (GPP)	[18]	G250.01	250 nodes, 662 edges		29
		G250.02	250 nodes, 1224 edges		114
		G250.04	250 nodes, 2566edges		357
		G250.08	250 nodes, 4842edges		828
Quadratic Assignment Problem (QAP)	[9]	tho40	40 facilities/locations		120258
		esc64a	64 facilities/locations		58
		sko72	72 facilities/locations		66256
		wil100	100 facilities/locations		48816
Generalised Assignment Problem (GAP)	[7]	gapA10-100	100 jobs, 5 agents	1360	
		gapA10-200	200 jobs, 10 agents	2623	
		gapA20-100	100 jobs, 5 agents	1158	
		gapA20-200	200 jobs, 10 agents	2339	
Bin Packing Problem (BIN)	[7]	bin3a1	500 items	198	
		bin3a2	500 items	201	
		bin3a3	500 items	202	
		bin3a4	500 items	204	
Traveling Salesman Problem (TSP)	[24]	st70	70 cities	675	
		kroA100	100 cities	21282	
		ch130	130 cities	6110	
		a280	280 cities	2579	

In this study, we follow the guidelines for reporting parallel experiments as outlined in [6]. The most common measure of effectiveness of a parallel algorithm is given by speedup. Speedup can be defined according to 7 [6]:

$$speedup = \frac{\text{Time to solve a problem with the fastest serial code on a specific parallel computer}}{\text{Time to solve the problem with the parallelcode } P \text{ processors on the same computer}} \quad (7)$$

From these data, speedup graphs are produced. In order to ascertain the effectiveness of our parallel code, the efficiency for each problem is computed 8.

$$efficiency = \frac{speedup}{P} \quad (8)$$

The parallel tabu search engine presented in this study is parameter driven. Determining appropriate parameter sets can be a time consuming exercise. Accordingly, we have chosen the most promising parameter sets given in [22, 23] for each problem type. The aim of this study is to test the parallel performance of the general TS solver. An extensive collection of results of the TS solver tested using various parameter settings may be found in [22, 23].

7 Computational Results

The TS engine is run with the best parameter settings obtained in [23]. Table 2 presents the best-received solution costs for each of the problems as well as the median sequential runtime. Table 3 shows the parallel speedup and efficiency gained by running the test problems.

Table 2. Best-received costs for the test problems.

Problem	Instance	Best-Known / Optimal Cost	Best Received Cost	Median Sequential Runtime
GPP	G250.01	29	32	288.5
	G250.02	114	117	1467
	G250.04	357	357	1366.5
	G250.08	828	830	1513
QAP	tho40	120258	120316	1061.5
	esc64a	58	58	23.5
	sko72	33158	33196	1430.5
	wil00	136522	136825	1495
GAP	gapA10-100	1360	1360	281
	gapA10-200	2623	2624	893
	gapA20-100	1158	1158	325
	gapA20-200	2339	2341	839.5
BIN	bin3a1	0	0	135
	bin3a2	0	4	40
	bin3a3	0	0	1330.5
	bin3a4	0	7	1256.5
TSP	st70	675	675	983.5
	kroA100	21282	21363	1015
	ch130	6110	6164	1143.5
	a280	2579	3527	1250.5

8 Conclusions

In this paper, a general parallel TS solver of COPs is proposed and implemented. This work is based on the list modelling system developed by [22, 23]. The parallel method used is based on a master-slave model in which the master controls the

Table 3. Parallel speedup and efficiency results. The first entry in each cell is speedup while the second is efficiency.

Problem	Instance	Processors	2	3	4	5	6	7	8	9	10	11	12	
Problem	Instance	Processors	2	3	4	5	6	7	8	9	10	11	12	
GPP	G250.01	1	0.63	0.95	1.7	2.43	3.12	3.72	4.42	5.22	5.71	6.2	6.71	7.18
		0.63	0.48	0.57	0.61	0.62	0.62	0.63	0.65	0.63	0.62	0.61	0.6	
	G250.02	0.65	1.02	1.79	2.66	3.32	3.96	4.69	5.55	6.23	6.65	7.02	7.82	
		0.65	0.51	0.6	0.67	0.66	0.66	0.67	0.69	0.69	0.67	0.64	0.65	
	G250.04	0.67	1.15	1.85	2.63	3.56	4.45	5.1	6.01	6.57	7.04	7.57	8.73	
		0.67	0.57	0.62	0.66	0.71	0.74	0.73	0.75	0.73	0.7	0.69	0.73	
	G250.08	0.62	1.1	1.88	2.88	3.69	4.39	5.08	5.74	6.46	7.19	8.07	8.17	
		0.62	0.55	0.63	0.72	0.74	0.73	0.73	0.72	0.72	0.72	0.73	0.68	
QAP	tho40	0.96	1.9	2.77	3.57	3.83	5.01	5.69	6.15	6.41	6.91	6.79	7.29	
		0.96	0.95	0.92	0.89	0.77	0.83	0.81	0.77	0.71	0.69	0.62	0.61	
	esc64a	0.95	1.61	2.32	3.06	3.69	4.4	4.96	5.4	5.54	6.19	6.58	6.85	
		0.95	0.81	0.77	0.76	0.74	0.73	0.71	0.67	0.62	0.62	0.6	0.57	
	sko72	0.98	1.95	2.9	3.9	4.83	5.65	6.46	7.53	8.44	8.97	10.07	10.93	
		0.98	0.97	0.97	0.97	0.97	0.94	0.92	0.94	0.94	0.9	0.92	0.91	
	wil100	0.99	1.96	2.94	3.89	4.88	5.83	6.78	7.68	8.63	9.43	10.44	11.24	
		0.99	0.98	0.98	0.97	0.98	0.97	0.97	0.96	0.96	0.94	0.95	0.94	
GAP	gapA10-100	0.97	1.42	2.15	2.69	3.3	3.52	4.01	4.16	4.56	4.22	4.77	4.46	
		0.97	0.71	0.72	0.67	0.66	0.59	0.57	0.52	0.51	0.42	0.43	0.37	
	gapA10-200	0.99	1.78	2.62	3.78	4.18	4.94	5.83	6.97	7.12	8.22	8.69	9.47	
		0.99	0.89	0.87	0.95	0.84	0.82	0.83	0.87	0.79	0.82	0.79	0.79	
	gapA20-100	0.97	1.59	2.56	2.94	3.36	3.78	4.16	4.56	4.72	5.15	5.03	5.86	
		0.97	0.8	0.85	0.74	0.67	0.63	0.59	0.57	0.52	0.52	0.46	0.49	
	gapA20-200	0.98	1.62	2.49	3.23	3.99	4.74	5.62	5.4	6	6.94	7.68	8.31	
		0.98	0.81	0.83	0.81	0.8	0.79	0.80	0.67	0.67	0.69	0.7	0.69	
BIN	bin3a1	0.99	1.71	2.45	3.16	3.84	4.65	5.2	6.02	6.59	7.4	7.89	8.57	
		0.99	0.85	0.82	0.79	0.77	0.78	0.74	0.75	0.73	0.74	0.72	0.71	
	bin3a2	0.99	1.78	2.55	3.29	4.06	4.88	5.51	6.32	7.09	7.77	8.42	9.07	
		0.99	0.89	0.85	0.82	0.81	0.81	0.79	0.79	0.79	0.78	0.77	0.76	
	bin3a3	0.99	1.71	2.48	3.20	3.91	4.65	5.36	6.01	6.73	7.68	8.15	8.79	
		0.99	0.85	0.83	0.8	0.78	0.78	0.77	0.75	0.75	0.77	0.74	0.73	
	bin3a4	0.99	1.74	2.46	3.28	4.01	4.72	5.41	6.13	6.75	7.76	8.19	8.68	
		0.99	0.87	0.82	0.82	0.8	0.79	0.77	0.77	0.75	0.78	0.75	0.72	
TSP	st70	0.99	1.76	2.58	3.62	4.4	5.2	5.79	6.1	7.04	7.82	8.13	8.69	
		0.99	0.88	0.86	0.91	0.88	0.87	0.83	0.76	0.78	0.78	0.74	0.72	
	kroA100	0.69	1.32	1.96	2.42	2.96	3.35	3.47	4.22	4.57	4.95	5.21	5.73	
		0.69	0.66	0.65	0.61	0.59	0.56	0.5	0.53	0.51	0.49	0.47	0.48	
	ch130	0.99	1.94	2.83	3.72	4.62	5.31	6.22	7.04	7.8	8.55	9.49	10.05	
		0.99	0.97	0.94	0.93	0.92	0.88	0.89	0.88	0.87	0.85	0.86	0.84	
	a280	0.88	1.78	2.57	3.37	4.18	5.05	5.83	6.53	7.47	8.11	8.95	9.77	
		0.88	0.89	0.86	0.84	0.84	0.84	0.83	0.82	0.83	0.81	0.81	0.82	

tabu search strategy while the slaves are used to evaluate the neighbours at each iteration. This approach was chosen as it is easy to apply across problem types and its performance has been reported to be scalable.

The performance of the parallel TS is generally dependent on the complexity of the objective function and the number of neighbours that are evaluated at each iteration. This is due to the parallelisation strategy adopted by this study. Parallel communication and housekeeping activities degrade the performance of the algorithm especially where the neighbourhood is small. This is also not helped by the fact that we use the strict version of speedup and efficiency in which the numerator is the fastest time for the sequential implementation rather than the time for running the parallel code on one processor. As the number of processors is increased, the time required to broadcast the neighbourhood division details and to process the incoming neighbours (sequential tasks) becomes large. Hence there is a reduction in efficiency as the number of processors increases. However the large QAPs generally record efficiencies above 90

As well as the above, parallel speedup and efficiency are adversely effected by the use of incremental cost expressions. This is because incremental cost expressions reduce the amount of computation required for the evaluation of the neighbourhood (i.e. the component of TS that has been parallelised). In order to demonstrate this, a problem instance that the parallel algorithm performed poorly on, gapA10-100, was run without using the incremental cost expression (Table 4). Using incremental cost expressions, at most 4 operations are required to compute the incremental cost, whereas a full cost expression requires $O(N)$ operations. Subsequently both speedup and efficiency are improved when the full cost function is used, though the performance of TS is substantially degraded. This is verified using the Mann-Whitney U test that compared efficiency with and without incremental cost expressions and gave $P=0.02$.

This work was undertaken as part of a larger project that aims to develop Application Specific Computers (ASCs) for solving COPs [1–4]. The benefit of these systems is that a specific hardware design has real time advantages over conventional workstations. The architectures of these computers will be designed to support a range of meta-heuristic search techniques through system reconfiguration using the list modelling system as the general representation scheme. As such, they will be based on Field Programmable Gate Array (FPGA) technology. FPGAs allow hardware to be reconfigured without any physical modification. This offers enormous potential because it should be possible to build a reconfigurable special purpose architecture, which can be used to solve a range of optimisation problems. It is envisaged that the ASC will be connected to a conventional workstation to provide the interface.

References

1. Abramson, D. (1992) "A Very High Speed Architecture to Support Simulated Annealing", IEEE Computer, pp. 27-34.
2. Abramson, D., de Silva, A., Randall, M. and Postula, A. (1995) "Special Purpose Computer Architectures for High Speed Optimisation", Proceedings of the Second

Table 4. Parallel speedup and efficiency for gapA10-100 with and without using incremental cost expressions.

Processors	With Incremental Cost Expressions Speedup	Efficiency	Time	Without Incremental Cost Expressions Speedup	Efficiency
1	0.97	0.97	281	0.99	0.99
2	1.43	0.71	190.98	1.78	0.89
3	2.15	0.71	126.81	2.52	0.84
4	2.69	0.67	100.97	3.27	0.82
5	3.30	0.66	82.49	3.97	0.79
6	3.52	0.59	77.35	4.64	0.77
7	4	0.57	67.85	5.53	0.79
8	4.16	0.52	65.46	5.88	0.74
9	4.57	0.51	59.64	6.55	0.73
10	4.22	0.42	64.43	6.77	0.68
11	4.77	0.43	57.07	7.23	0.66
12	4.46	0.37	61.01	7.46	0.62

Australasian Conference on Parallel and Real Time Systems Conference, pp. 13 - 20.

3. Abramson, D., Logothetis, P., Randall, M. and Postula, A. (1997) "Application Specific Computers for Combinatorial Optimisation", The Australian Computer Architecture Workshop, Sydney, Australia, February.
4. Abramson, D., Logothetis, P., Randall, M. and Postula, A. (1998) "A Tail Of 2n Cities: Performing Combinatorial Optimisation Using Linked Lists On Special Purpose Computers", Keynote address, Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, February, Gippsland. pp. 17-45.
5. Badeau, P., Gedreau, M., Guerin, F., Potvin, J. and Taillard, E. (1995) "A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows", Transportation Research-C, **5**, pp. 109-122.
6. Barr, R. and Hickman, B. (1993) "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures and Experts' Opinions", ORSA Journal on Computing, **5**, pp. 2-18.
7. Beasley, J. (1990) "OR-library: Distributing Test Problems by Electronic Mail", Journal of the Operational Research Society, **41**, pp. 1069-1072.
8. Brooke, A., Kendrick, D., Meeraus, A. and Raman, R. (1997) GAMS Language Guide, Washington: GAMS Development Corporation, 305 pages.
9. Burkard, R., Karisch, S. and Rendl, F. (1997) "QAPLIB - A Quadratic Assignment Problem Library", Journal of Global Optimization, **10**, pp. 391-403.
10. Chu, P. and Beasley, J. (1995) "A Genetic Algorithm for the Generalised Assignment Problem", Computers and Operations Research, **24**, pp. 17-23.
11. Crainic, T., Toulouse, M. and Gendreau, M. (1997) "Toward a Taxonomy of Parallel Tabu Search Heuristics", INFORMS Journal on Computing, **9**, pp. 61-71.
12. De Falco, I., Del Balio, R. and Tarantino, E. (1996) "Solving the Mapping Problem by Parallel Tabu Search", Technical Report - Istituto per la Ricerca sui Sistemi Informatici Paralli, Italy.

13. Garcia, B., Potvin, J. and Rousseau, J. (1994) "A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints", *Computers and Operations Research*, **21**, pp. 1025-1033.
14. Glover, F. (1989) "Tabu Search - Part I", *ORSA Journal on Computing*, **1**, pp. 190-206.
15. Glover, F. (1990) "Tabu Search - Part II", *ORSA Journal on Computing*, **2**, pp. 4-32.
16. Glover, F. and Laguna, M. (1997) *Tabu Search*, Boston: Kluwer Academic Publishers, 402 pages.
17. Hertz, A., Taillard, E. and de Werra, D. (1997) "Tabu Search", In Aarts, E. and Lenstra, J. (eds.), *Local Search in Combinatorial Optimization*, New York: Wiley, pp. 121-136.
18. Johnson, D., Aragon, C., McGeogh, L. and Scheveon, C. (1991) "Optimization by Simulated Annealing: An Experimental Evaluation Pt I, Graph Partitioning", *Operations Research*, **37**, pp. 865-892.
19. Mavridou, T., Pardalos, P., Pitsoulis, L. and Resende, M. (1995) "Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP", *Proceedings of Parallel Algorithms for Irregularly Structured Problems*.
20. Osman, I. (1993) "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem", *Annals of Operations Research*, **41**, pp. 421-451.
21. Osman, I. (1995) "Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches", *Operations Research Spektrum*, **17**, pp. 211-225.
22. Randall, M. (1998) "A General Modelling System and Solver for Combinatorial Optimisation Problems", PhD thesis, School of Environmental and Applied Science, Griffith University.
23. Randall, M. and Abramson, D. (1999) "A General Meta-heuristic Based Solver for Combinatorial Optimisation Problems", Technical Report **99-01**, School of Information Technology, Bond University, Submitted to the *Journal of Computational Optimization and Applications*.
24. Reinelt, G. (1991) "TSPLIB - a Traveling Salesman Problem Library", *ORSA Journal on Computing*, **3**, pp. 376-384.
25. Taillard, E. (1991) "Robust Taboo Search for the Quadratic Assignment Problem", *Parallel Computing*, **17**, pp. 443-455.
26. Taillard, E. (1993) "Parallel Iterative Search Methods for Vehicle Routing Problems", *Networks*, **23**, pp. 661-673.
27. van Laarhoven, P. and Aarts, E. (1987) *Simulated Annealing: Theory and Applications*, Dordrecht: D. Reidel Publishing Company, 186 pages.