Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 185 (2008) 1192–1212

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

The development of a multi-objective Tabu Search algorithm for continuous optimisation problems

D.M. Jaeggi *, G.T. Parks, T. Kipouros, P.J. Clarkson

Engineering Design Centre, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK

Received 1 July 2005; accepted 1 June 2006
Available online 25 October 2006

Abstract

While there have been many adaptations of some of the more popular meta-heuristics for continuous multi-objective optimisation problems, Tabu Search has received relatively little attention, despite its suitability and effectiveness on a number of real-world design optimisation problems. In this paper we present an adaptation of a single-objective Tabu Search algorithm for multiple objectives. Further, inspired by path relinking strategies common in discrete optimisation problems, we enhance our algorithm to allow it to handle problems with large numbers of design variables. This is achieved by a novel parameter selection strategy that, unlike a full parametric analysis, avoids the use of objective function evaluations, thus keeping the overall computational cost of the procedure to a minimum. We assess the performance of our two Tabu Search variants on a range of standard test functions and compare it to a leading multi-objective Genetic Algorithm, NSGA-II. The path relinking-inspired parameter selection scheme gives a clear performance improvement over the basic multi-objective Tabu Search adaptation and both variants perform comparably with the NSGA-II.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Tabu Search; Global optimisation; Genetic algorithms; Multiple criteria analysis; Meta-heuristics

1. Introduction

Meta-heuristic optimisation techniques have proved effective in solving complex, real-world optimisation problems with many local minima, problems to which traditional, gradient-based methods are ill suited. Some significant methods to have emerged in recent years are Simulated Annealing (SA), Genetic Algorithms (GA), Evolution Strategies (ES) and Tabu Search (TS).

With the realisation that most real-world design problems involve compromises between conflicting objectives, efforts have been directed towards developing multi-objective optimisation algorithms. The first multi-objective GA was developed in 1985 and research in this field has been very active [8]. Similarly, multi-objective SA [30] and ES [26] algorithms have also been developed . However, very little work has been done on multi-objective TS algorithms. Jones et al. [22] surveyed 115 articles concerned with multi-objective meta-heuristics for both discrete and continuous problems. They found 70% of the articles used GA or ES as the primary meta-heuristic, 24% SA and 6% TS.

* Corresponding author.
E-mail address: dmj22@cam.ac.uk (D.M. Jaeggi).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.06.048

In this paper we present two multi-objective TS algorithms. The first, previously presented algorithm is a straightforward adaptation of a single-objective TS algorithm for continuous problems. The second algorithm contains significant enhancements, including a novel design variable selection scheme (to improve local search efficiency) based on path relinking strategies common in discrete optimisation algorithms.

The motivation for these developments in the TS algorithm is our interest in aerodynamic shape optimisation problems. Such problems have high computational demands, are highly constrained, and cannot be solved by gradient-based methods due to the lack of gradient information and the large number of local minima. Previous research on a single-objective aerodynamic problem has found TS to be a particularly effective optimisation algorithm for this application domain [17] and we believe that this will carry over into multi-objective problems in the same domain and also in any domain where the design space is highly constrained [20]. Similar research in other fields has also suggested TS as a suitable optimisation algorithm for these kinds of problems [5].

The development of the design variable selection scheme was motivated by the authors' belief that real-world optimisation problems will increasingly have larger numbers of design variables. Good parameterisation schemes for aerodynamic shape optimisation problems – as shown by Harvey [17], Kellar [23], Duvigneau and Visonneau [10] and Gaiddon et al. [13] – tend to produce optimisation landscapes with many variables.

Of particular interest is the parameterisation scheme – Kellar's observation that, given a perfect optimisation algorithm and a zero cost objective function evaluation, the effectiveness of an optimisation algorithm is solely dependent on the range of possible designs it can generate is particularly pertinent. Kellar showed that a good and highly flexible parameterisation scheme for the shape optimisation of a Formula 1 racing car front wing could have of the order of 1000 design variables [23]. The shape optimisation of a Boeing 747 wing performed by Jameson [21] required 90 design variables.

Of course, the effectiveness of optimisation in a real-world situation will depend on the interplay between all three components – the optimisation algorithm, the parameterisation scheme and the underlying simulation generating the objective function values – but it is clear that optimisation algorithms in the future will have to cope with greater numbers of design variables.

Furthermore, in the same work Kellar performed a parametric study on a number of optimisation problems and showed that the sensitivity of an objective function to each design variable was not consistent across the design space. Specifically, for each design variable, there were regions in the design space where that variable had little effect on the objective function and regions where it had a great effect. Thus, he showed that optimising on a dynamically changing subset of design variables, rather than the entire set of variables, led to a great optimiser performance improvement. However, it is important to note that this is only possible using a local search method (such as TS) as the design variable sensitivities are local themselves.

Ideally, one could devise an optimisation scheme that employed a full parametric sensitivity analysis as part of the ongoing search. This, however, would be unfeasibly expensive when one considers the number of function evaluations required. Kellar circumvented this problem by performing cheap geometric comparisons with a dynamic reference set of known good solutions. This is analogous to path relinking [15] employed in discrete optimisation strategies. Inspired by Kellar's work, we modified his strategy so that it could be applied to any continuous optimisation problem and incorporated it into our multi-objective TS algorithm.

In the remainder of this paper, we will describe these developments in more detail and present results of a comparison between our two TS variants and a leading multi-objective Genetic Algorithm, NSGA-II [9]. Although comparisons between one of these TS algorithms and NSGA-II have been previously published [19,20], this paper describes the significant enhancements made in the new TS variant and provides a far more rigourous performance assessment of all three algorithms.

2. Background

2.1. Multi-objective optimisation

The goal of single-objective optimisation is to find a vector $\bar{x}$ such that $f(\bar{x})$ attains its minimal value, its optimum. The vector $\bar{x}$ is called a vector of design variables and the function $f(\bar{x})$ is called an objective function. A point $\bar{x}_1$ in design space can be considered "better" than another point $\bar{x}_2$ if $f(\bar{x}_1) < f(\bar{x}_2)$. Thus, single-objective optimisation has only one optimal solution cost (although there is the possibility that this may be realised by multiple design vectors).

In multi-objective optimisation, we are tasked with minimising not just one objective function, but n objective functions $f_1(\bar{x}), \ldots, f_n(\bar{x})$, where $n \geqslant 2$. The solution to this problem is more complex than the single-objective case, and the idea of Pareto-dominance must be introduced to be able to visualise it. Consider first an objective function vector $\overline{F}(\bar{x})$, where $\overline{F}(\bar{x}) = \{f_1(\bar{x}), \ldots, f_n(\bar{x})\}$. A point $\bar{x}_1$, with an objective function vector $\overline{F}_1$, is said to *dominate* point $\bar{x}_2$, with an objective function vector $\overline{F}_2$, if no component of $\overline{F}_1$ is greater than its corresponding component in $\overline{F}_2$, and at least one component is smaller. Similarly, $\bar{x}_1$ can be said to be Pareto-equivalent to $\bar{x}_2$ if some components of $\overline{F}_1$ are greater than $\overline{F}_2$ and some are smaller. Pareto-equivalent points represent a trade-off between the objective functions, and it is impossible to say that one point is "better" than another Pareto-equivalent point without introducing preferences or relative weighting of the objectives.

Thus, the solution to a multi-objective optimisation problem is a set of design vectors which are not dominated by any other vector, and which are Pareto-equivalent to each other. This set is known as the Pareto-optimal set.

2.2. Existing multi-objective Tabu Search algorithms

There are two commonly used approaches to solving a multi-objective optimisation problem [30]. The first reduces the multiple objectives to a single objective by generating a composite objective function, usually from a weighted sum of the objectives. This composite objective function can be optimised using existing single-objective optimisers. However, the weights must be pre-set, and the solution to this problem will be a single vector of design variables rather than the entire Pareto-optimal set. This can have undesirable consequences: setting the weights implicitly introduces the designer's preconceptions about the relative trade-off between objectives. Real-world problems can produce surprising Pareto-optimal sets which may profoundly affect design decisions, and the potential to generate novel designs is a key benefit of optimisation [30]. The TS algorithms reviewed by Jones et al. [22] use this approach to solve the multi-objective problem.

The second approach to solving the multi-objective problem is to search directly for the entire Pareto-optimal set. This can be achieved in a number of ways and requires modification to existing single-objective algorithms.

The authors know of only two other attempts to produce a multi-objective TS algorithm which finds multiple Pareto-optimal solutions to a continuous optimisation problem in a single run. Hansen's algorithm [16] is an extension of the composite objective approach: his algorithm performs a number of composite objective Tabu searches in parallel. Each search has a different and dynamically updated set of weights, and in this way the search can be driven to explore the entire Pareto front. This algorithm, although a good implementation of TS, suffers the problems common to all weighted-sum approaches: for problems with concave Pareto fronts, there may be regions of the front that are not defined by a combination of weights, and conversely certain combinations of weights represent two points on the front. Thus, this algorithm may not adequately locate the entire Pareto-optimal set.

Baykasoglu et al. [2] developed a TS algorithm combining a downhill local search with an *intensification memory* (IM) to store non-dominated points that were not selected in the search. When the search fails to find a downhill move, a point from the IM is selected instead. When the IM is empty and all search paths exhausted, the algorithm stops. This cannot be considered a true TS algorithm: in restricting the search to only downhill moves its originators reject one of the basic tenets of TS, that "a bad strategic choice can yield more information than a good random choice" [15]. Also, the lack of any diversification strategy renders the algorithm incomplete and merely an elaborate local search algorithm.

For combinatorial optimisation problems, Caballero et al. [4] developed a novel TS based algorithm with a 2-phase approach: the first phase consisted of a local Tabu Search algorithm to generate a list of locally Pareto-equivalent solutions (those solutions which are non-dominated with respect to their immediate neighbours); the second phase involved an intensification procedure using these solutions, which employed a path-relinking strategy. While the structure of their algorithm is quite different, with elements being tailored to combinatorial optimisation problems, it is a good implementation and it contains some interesting ideas.

Jaeggi et al. developed the first multi-objective TS variant presented in this paper and executed a performance comparison on a set of unconstrained test functions [19]. The constraint handling approach and the performance of the algorithm on constrained optimisation problems was also presented in [20].

2.3. Path relinking

Path relinking strategies are quite common in the discrete optimisation field. They were developed for use in TS [15] and Scatter Search [14] algorithms, and have also been used in Greedy Randomised Adaptive Search Procedures [1].

The underlying principle is that knowledge of the path (or direction) in design space can be used to guide the search along that path, given that it may be a better search direction than a random one, in the hope that it will yield further good points.

A path can be determined by considering an initial point and a guiding (or reference) point. The reference point is a known good solution in the search. In essence, considering the points as vectors of design variables, the path between the points is a linear combination of their design variables and, as such, retains information about the reference point, leading ideally to further good points [23].

3. Multi-objective Tabu Search adaptation

In this paper, we present two multi-objective TS algorithms. The first, previously presented algorithm [19,20] we call MOTS (Multi-Objective Tabu Search). The second algorithm, which contains the newly developed design variable selection scheme, we call PRMOTS (Path Relinking Multi-Objective Tabu Search). The fundamental basis for both algorithms is the same and is presented in this section. The two algorithms differ in their local search pattern, a key component of TS and an area in which significant performance enhancements can be made, and this will be described in Section 4.

3.1. Algorithm overview

The single-objective TS implementation of Connor and Tilley [6] is used as a starting point for our multi-objective variants. This uses a Hooke and Jeeves (H&J) local search algorithm (designed for continuous optimisation problems) [18] coupled with short, medium and long term memories to implement search intensification and diversification as prescribed by Glover and Laguna [15].

TS operates in a sequential, iterative manner: the search starts at a given point and the algorithm selects a new point in the search space to be the next current point. The basic search pattern in Connor and Tilley's implementation is the H&J search.

Recently visited points are stored in the *short term memory* (STM) and are Tabu – the search is not allowed to revisit these points. Optimal or near-optimal points are stored in the *medium term memory* (MTM) and are used for intensification, focusing the search on areas of the search space with known good objective function values. The *long term memory* (LTM) records the regions of the search space which have been explored, and is used on diversification, directing the search to regions which are under-explored. This is achieved by dividing each design variable into *n_regions* regions and counting the number of solutions evaluated in those regions. A local iteration counter *i_local* is used and reset upon a successful addition to the MTM. When *i_local* reaches user-specified values, the algorithm will diversify or intensify the search, or reduce the search step size and restart the search from the best solution found.

Thus, TS combines a systematic local search with a stochastic element and an intelligent coverage of the entire search space. Our multi-objective TS implementation of Connor and Tilley's algorithm [6] is modified in the following areas: search point comparison; the H&J move; optimal point archiving and the MTM; search intensification and restart strategy. These modifications are described below, along with some further improvements. A flow diagram of this algorithm is shown in Fig. 1.

A pseudocode listing for the main TS procedure is presented in Algorithm 1, along with subroutines presented in Algorithms 2–11.

Algorithm 1

```
TabuSearch
 1: $STM$ = new $Container$
 2: $MTM$ = new $Container$
 3: $LTM$ = new $Container$
 4: $IM$ = new $Container$
 5: $iIter = 0$
 6: $iLocal = 0$
 7: while $stoppingCriteriaNotMet()$ do
 8:    if $nextMoveHookeJeeves$ then
 9:       $HookeJeevesMove()$
10:    else
11:       $success = PatternMove()$
12:       if $success$ then
13:          $nextMoveHookeJeeves = 1$
```

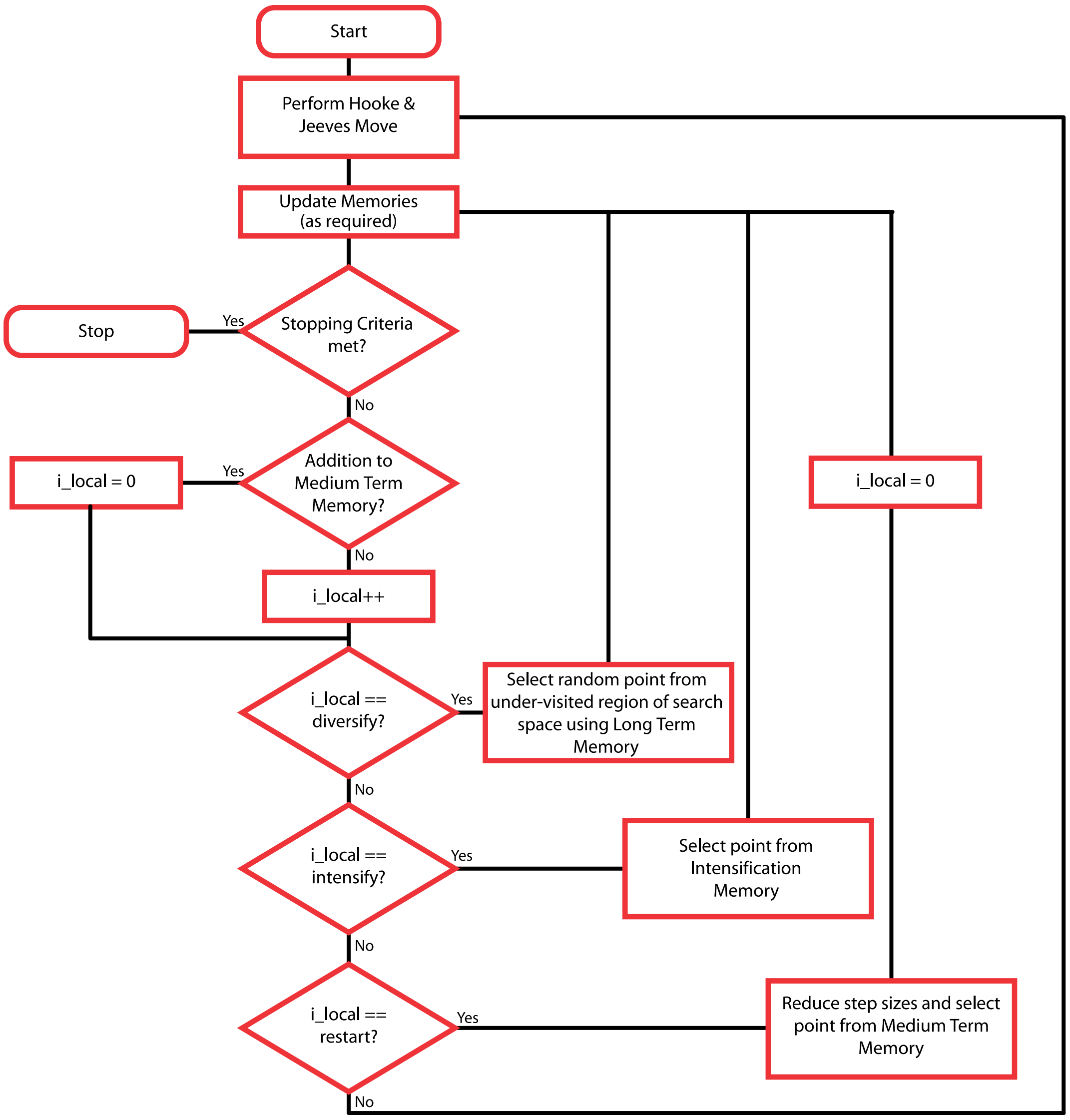
Start
Perform Hooke & Jeeves Move
Update Memories (as required)
Stop
Yes
Stopping Criteria met?
No
i_local = 0
Yes
Addition to Medium Term Memory?
No
i_local++
i_local = 0
i_local == diversify?
Yes
Select random point from under-visited region of search space using Long Term Memory
No
i_local == intensify?
Yes
Select point from Intensification Memory
No
i_local == restart?
Yes
Reduce step sizes and select point from Medium Term Memory
No

Fig. 1. Flow diagram of the multi-objective Tabu Search algorithm.

```
14:       else
15:          $HookeJeevesMove()$
16:       end if
17:    end if
18:    $UpdateMemories()$
19:    if $iLocal == diversify$ then
20:       $DiversifyMove()$
21:       $nextMoveHookeJeeves = 1$
22:       goto 18
23:    else if $iLocal == intensify$ then
24:       $IntensifyMove()$
25:       $nextMoveHookeJeeves = 1$
26:       goto 18
27:    else if $iLocal == reduce$ then
28:       $ReduceMove()$
29:       $nextMoveHookeJeeves = 1$
30:       goto 18
31:    end if
32: end while
```

Algorithm 2

```
$dominates(A,B)$
 1: for each $objective$ do
 2:    if $objectiveValue\ A > objectiveValue\ B$ then
 3:       $nGreater$++
 4:    else if $objectiveValue\ A < objectiveValue\ B$
       then
 5:         $nLess$++
 6:     end if
 7: end for
 8: if $nGreater > 0$ and $nLess == 0$ then
 9:     return $-1$
10: else if $nLess > 0$ and $nGreater == 0$ then
11:     return 1
12: else
13:     return 0
14: end if
```

Algorithm 3

```
removeDominatedPoints(container)
 1: for each pointA in container do
 2:     for each pointB in container do
 3:         if not pointA==pointB then
 4:             if dominates(pointA,pointB) then
 5:                 removePoint(pointB,container)
 6:             end if
 7:         end if
 8:     end for
 9: end for
```

Algorithm 4

```
isTabu(point)
 1: for each tabuPoint in STM do
 2:     if point==tabuPoint then
 3:         return 1
 4:     end if
 5: end for
 6: return 0
```

Algorithm 5

```
addIfNotDominated(newPoint,container)
 1: for each existingPoint in container do
 2:     if dominates(existingPoint,newPoint) then
 3:         return 0
 4:     end if
 5: end for
 6: add(newPoint,container)
 7: return 1
```

Algorithm 6

```
HookeJeevesMove
 1: currentPoint = points(iIter)
 2: bestPoints = new Container
 3: for each designVariable do
 4:     newPoint = currentPoint + stepSizes(designVariable)
 5:     if isNotTabu(newPoint) and isNotInvalid(newPoint) then
 6:         addIfNotDominated(newPoint,bestPoints)
 7:         removeDominatedPoints(bestPoints)
 8:     end if
 9:     newPoint = currentPoint − stepSizes(designVariable)
10:     if isNotTabu(newPoint) and isNotInvalid(newPoint) then
11:         addIfNotDominated(newPoint,bestPoints)
12:         removeDominatedPoints(bestPoints)
13:     end if
14: end for
15: iIter++
16: iLocal++
17: nextPoint = selectRandom(bestPoints)
18: points(iIter) = nextPoint
19: removePoint(nextPoint,bestPoints)
20: for each remainingPoint in bestPoints do
21:     addIfNotDominated(remainingPoint,IM)
22: end for
23: removeDominatedPoints(IM)
24: nextMoveHookeJeeves = 0
```

Algorithm 7

```
PatternMove
 1: currentPoint = points(iIter)
 2: lastPoint = points(iIter − 1)
 3: lastMove = currentPoint − lastPoint
 4: newPoint = currentPoint + lastMove
 5: calculateObjectives(newPoint)
 6: if dominates(newPoint,currentPoint) then
 7:     iIter++
 8:     iLocal++
 9:     points(iIter) = newPoint
10:     return 1
11: else
12:     return 0
13: end if
```

Algorithm 8

```
UpdateMemories
 1: currentPoint = points(iIter)
 2: pushFront(currentPoint,STM)
 3: popBack(STM)
 4: success = addIfNotDominated(currentPoint,MTM)
```

```
5: if $success$ then
6:     $removeDominatedPoints(MTM)$
7:     $iLocal = 0$
8: end if
9: $add(currentPoint, LTM)$
```

Algorithm 9

DiversifyMove

```
1: $regionCount = $ new $array(nRegions * nVar)$
2: for each $point$ in $LTM$ do
3:     for each $region$ do
4:         if $point$ is in $region$ then
5:             $regionCount(region)++$
6:         end if
7:     end for
8: end for
9: $newRegion = min(regionCount)$
10: $newPoint = selectRandomPointFromRegion$
    $(newRegion)$
11: if $isTabu(newPoint)$ or $isNotValid(newPoint)$
     then
12:     $DiversifyMove()$
13: end if
14: $iIter++$
15: $iLocal++$
16: $points(iIter) = newPoint$
```

Algorithm 10

IntensifyMove

```
1: $newPoint = selectRandom(IM)$
2: if $isTabu(newPoint)$ then
3:     $IntensifyMove()$
4: end if
5: $iIter++$
6: $iLocal++$
7: $points(iIter) = newPoint$
8: $removePoint(newPoint, IM)$
```

Algorithm 11

ReduceMove

```
1: $newPoint = selectRandom(MTM)$
2: if $isTabu(newPoint)$
3:     $ReduceMove()$
4: end if
5: $reduce(stepSizes)$
6: $iIter++$
7: $iLocal = 0$
8: $points(iIter) = newPoint$
```

3.2. Search point comparison

In a single-objective optimisation problem, points may be compared using the operators ==, > and < acting on the objective function values for those points. Similarly, points in a multi-objective problem can be compared in the same way (thus preserving the logic of the single-objective algorithm) by using the concepts of Pareto-equivalence (==) and dominance (> and <), described in Section 2.1.

3.3. The Hooke and Jeeves move

At each iteration, a H&J move is made. $2n\_var$ new points are generated by incrementing and decrementing each of the n_var design variables by a given step around the current point. The objective functions for each new point are evaluated and, as long as the point is neither Tabu (i.e., not a member of the STM) nor violates any constraints, it is considered as a candidate for the next point in the search.

In the single-objective TS algorithm, these candidates are sorted and the point with the lowest objective is chosen as the next point. A similar logic can be applied to the multi-objective case: however, the possibility of multiple points being Pareto-equivalent and optimal must be allowed for.

This is achieved by classifying each candidate point according to its domination or Pareto-equivalence to the current point. If there is a single dominating point, it is automatically accepted as the next point. If there are multiple dominating points, the dominated points within that group are removed and one is selected at random from those remaining. The other points become candidates for intensification (discussed below). If there are no dominating points, the same procedure is applied to those candidate points which are Pareto-equivalent to the current point. If there are no Pareto-equivalent points, a dominated point is selected in the same fashion. Thus, our strategy accepts both downhill and uphill moves – the next point is simply the “best” allowed point (or one of the Pareto-equivalent best points) selected from the candidate solutions.

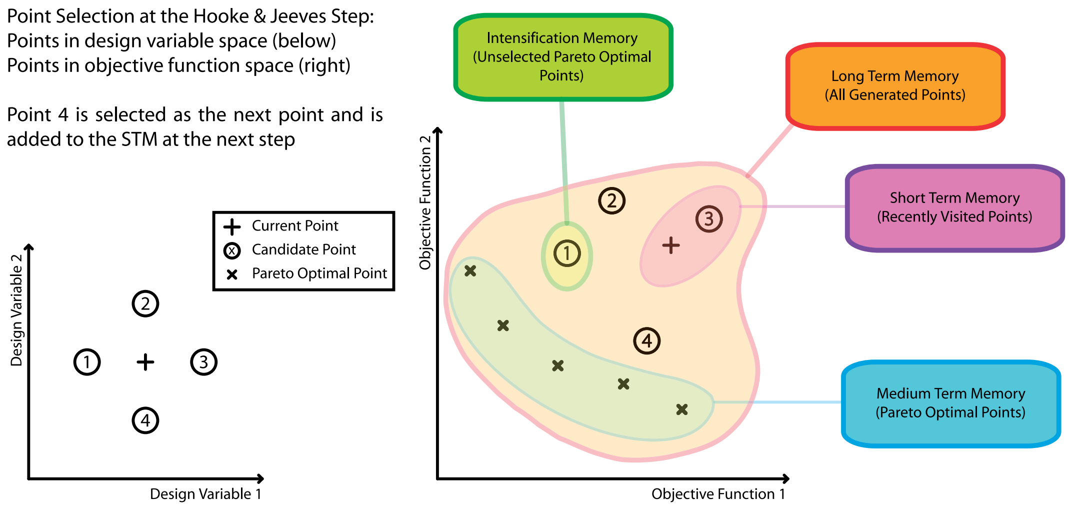
Point Selection at the Hooke & Jeeves Step:
Points in design variable space (below)
Points in objective function space (right)
Point 4 is selected as the next point and is added to the STM at the next step
Current Point
Candidate Point
Pareto Optimal Point
Design Variable 2
Design Variable 1
Objective Function 2
Objective Function 1
Intensification Memory (Unselected Pareto Optimal Points)
Long Term Memory (All Generated Points)
Short Term Memory (Recently Visited Points)
Medium Term Memory (Pareto Optimal Points)
1
2
3
4

Fig. 2. Point selection for the Hooke & Jeeves move and Tabu Search memories.

In addition, a *pattern move* strategy is implemented in the same way as Connor and Tilley. Before every second H&J move, the previous move is repeated. This new point is compared to the current point, and, if it dominates it, is accepted as the next point; if not, the standard H&J move is made. In this way, the search may be accelerated along known downhill directions. The basic search pattern is shown in Fig. 2.

3.4. Optimal point archiving and the medium term memory

In Connor and Tilley's single-objective TS, the MTM is a bounded, sorted list of near-optimal solutions. As the concept of a single optimal point does not exist in multi-objective optimisation (see Section 2.1), we replace the MTM in our multi-objective TS variant by an unbounded set of non-dominated solutions produced by the search. As new points are evaluated, they become candidates for addition to this set. Thus, the MTM represents the Pareto-optimal set for the problem at that stage in the search.

An unbounded MTM was chosen for two main reasons. First, for continuous problems the ideal Pareto set is infinite and an unbounded set is a logical discretisation of this concept. While there are legitimate concerns regarding the presentation of a large number of Pareto-optimal solutions to a designer as the result of an optimisation run, we feel it would be better to deal with these issues at the post-processing stage. It is worth pointing out that some GAs use bounded archives of optimal points because this is consistent with the use of fixed size populations in those algorithms.

Our second justification for the use of an unbounded set is entirely pragmatic: typically the number of solutions stored in the MTM is low. For example, for the problems presented in this paper, the MTM size rarely exceeded 150 – one would expect real-world problems run with a lower number of function evaluations to have smaller MTMs. For an MTM of this kind of size, the memory and run-time overhead is low.

3.5. Intensification and restart strategy

The original single-objective TS produced intensification points by using the MTM to generate points in the neighbourhood of good solutions. Although the replacement of the MTM by a Pareto-optimal set of solutions allows us to use a variant of this strategy, a feature of multi-objective optimisation suggests an alternative strategy, similar to that used by Baykasoglu et al. [2].

A multi-objective H&J iteration may produce multiple Pareto-optimal points (see Fig. 2). As only one point may be selected as the next point, it seems wasteful to discard the other points. Therefore, we incorporate an intensification memory into our algorithm. This is a set of Pareto-equivalent points; at each H&J step, points which dominate the current solution, but are not selected as the next point (of which there can be only one), are considered as candidates for addition to the set. At search intensification, a point is chosen randomly from the IM. The IM is continuously updated and points which become dominated by the addition of a new point are removed. Thus, the IM should always contain points which are on, or near to, the current Pareto-optimal front (stored in the MTM). Fig. 2 shows the relationship between the various TS memories.

The single-objective TS restart strategy returns the search to the current best point in the MTM. As the MTM is now a set of Pareto-optimal points, we simply select one point at random from the set. More intelligent restart strategies are possible [30] and are under investigation.

3.6. Constraint handling

We employ a very simple constraint handling strategy: any point which violates any constraint is deemed to be Tabu and the search is not allowed to visit that point. Thus, accepted solutions are limited to feasible space. We refer to this as a binary constraint – solutions are either feasible or not and there is no need to quantify the extent of constraint violation.

A common constraint handling approach is that using penalty functions [9]. Here, the extent of constraint violation must be quantified and this value is used as a surrogate objective to drive the search into feasible regions.

Our algorithm has the flexibility to use both kinds of constraint handling as the introduction of penalty functions or modified dominance relations [9] is trivial. We believe that this approach gives our algorithm significant power in tackling real-world problems, particularly where complex simulations are being used to evaluate objective functions. Here, there is a possibility that a simulation may not converge and thus a solution may not be feasible, with no further information. Indeed, it is the authors' experience that the attainment of a fully converged CFD solution for novel geometries is a non-trivial task and there are frequent simulation failures. These failures tend not to be an issue of software quality, rather they reflect the overstepping of boundaries in the physical model being used. These issues are also reported by researchers in other fields [5]. A further benefit to running our algorithm on highly constrained problems is the local search pattern at the heart of TS, which reduces the likelihood of generating infeasible solutions.

3.7. Parallelisation strategy

Any optimisation procedure that forms part of a real-world design cycle must be able to complete in a reasonable time frame. Parallel processing offers a large potential speed-up; an optimisation algorithm should ideally be designed with this in mind. Our multi-objective TS algorithm is parallelised by means of functional decomposition. At each H&J move, the required objective function evaluations are computed in parallel. This gives the potential for near-linear speed-up on parallel computers with up to $2n\_var$ processors (this being the maximum number of required function evaluations at each iteration, determined by the H&J search).

4. Local search efficiency improvements

The H&J local search strategy requires roughly $2n\_var$ solution evaluations (allowing for points that are Tabu or violate constraints) at each step. This is the most computationally expensive part of the optimisation procedure – assuming that objective function evaluations are expensive compared to the optimisation algorithm logic, a reasonable assumption for most real-world problems – and there is considerable scope for efficiency savings at this step.

Our two TS variants are distinguished by the way in which they attempt to improve the search efficiency at this step. The original MOTS algorithm incorporates an element of random sampling with the aim of finding a "downhill" move (defined by Pareto-dominance) in fewer objective function evaluations. The PRMOTS algorithm restricts the dimensionality of the local search space by only using a set of design variables that are estimated to have the greatest impact on the objective function values. These two approaches are described in detail below.

4.1. Improved local search space sampling – MOTS

Following the H&J search pattern procedure, we generate the $2n\_var$ new points, remove those that are Tabu, but only evaluate $n\_sample \leqslant 2n\_var$ points from those that remain, selecting randomly to avoid introducing any directional bias. If one of these points dominates the current point, it is automatically accepted as the next point. If more than one point dominates the current point, a non-dominated point from these is randomly selected. If no points dominate the current point, a further *n_sample* points (or less, depending on the number of points remaining) are sampled randomly and the comparison is repeated. If all the feasible, non-Tabu points have been sampled without finding a point that dominates the current solution, the standard selection procedure is employed, as described in Section 3.3.

4.2. Design variable selection scheme – PRMOTS

The search space sampling in MOTS reduces the number of potential evaluations at each H&J step by introducing random sampling. PRMOTS attempts to reduce the number of evaluations in a more systematic way; PRMOTS allows some design variables to be "dormant" and others to be "active" in the search, based on their estimated likely impact on the objective function values. Thus, the dimensionality of the problem is temporarily reduced.

Ideally, one would perform a full parametric analysis to determine the full sensitivities of the objective functions to the design variables, and select the active variables according to these sensitivities. However, for a problem with a large number of design variables and expensive objective functions, this procedure would be prohibitively costly. Instead, our scheme uses the concept of a path in design space as a surrogate for the likely impact on the objective functions, allowing quasi-sensitivities to be calculated at near-zero cost.

After every *select_interval* TS iterations, variable selection takes place as follows. Based on the H&J search pattern, we generate the 2*n_var* new points. However, we now keep all the points without removing those that are Tabu and *without making any function evaluations*. We then assess the potential for improvement in the objective functions for each design variable in the following way. Each design variable has two new points associated with it, for which the H&J step has incremented or decremented that variable. Taking the current set of Pareto-optimal points (stored in the MTM) as the reference set, the Euclidean distance in design space between these two points and every point in the reference set is calculated. The smallest of these distances is saved, and the design variables are ranked according to this distance. The design variables with the *n_selected* smallest distances are set to be active for the next *select_interval* iterations, while the other variables are dormant and are kept fixed at their current values, until the procedure is repeated again. The calculation of the distances used to rank the design variables is shown graphically in Fig. 3.

In essence, this selection scheme is only allowing the optimiser to vary those variables that will progress the search from current point to the reference set in the fastest possible way. This assessment is based solely on a measure of the Euclidean distance in design space, which is used as a surrogate mea-

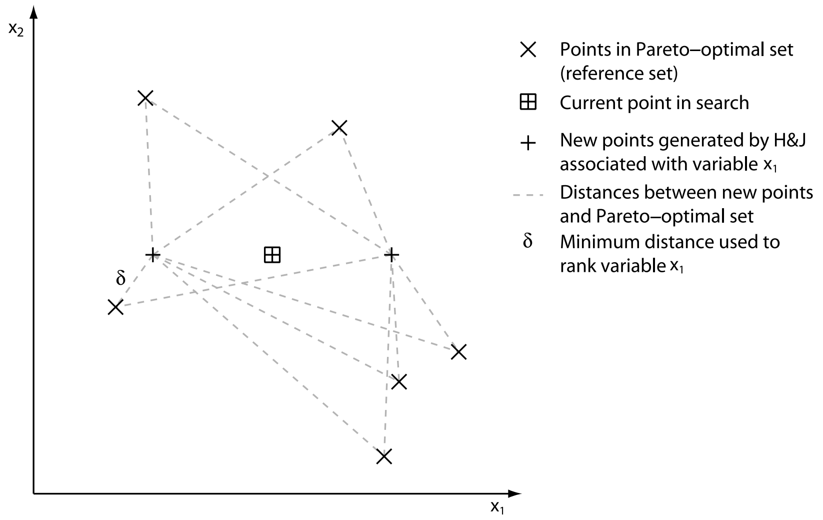
x_2
x_1
δ
Points in Pareto–optimal set (reference set)
Current point in search
New points generated by H&J associated with variable x_1
Distances between new points and Pareto–optimal set
Minimum distance used to rank variable x_1

Fig. 3. Graphical representation of the parameter selection scheme – calculation of the minimum Euclidean distance associated with variable x_1 to the reference set.

sure of the scale of potential improvement – it does not involve any evaluation of the objective functions.

5. Testing procedure

In order to assess the effectiveness of our two TS algorithms, we undertook a rigorous testing procedure on a set of benchmark test problems. There were two aims to this exercise: to assess the performance benefit of the design variable selection scheme; to compare the performance of both TS variants to a leading multi-objective optimisation algorithm.

5.1. Test functions

The test functions used in this study are from the ZDT family of problems [8] and are described in detail in Table 1. They have $10 \leqslant n\_var \leqslant 30$ and the number of objectives $n\_obj = 2$, and they are designed to have Pareto-optimal sets which are difficult to locate accurately by means of an optimisation algorithm. They have some shortcomings, namely that the locations of the Pareto-optimal sets in design space form continuous regions [28], and their relevance and applicability to real-world problems is debatable. However, they are easy to implement, well studied, fast to compute and provide a useful basis for assessing optimiser performance.

5.2. Benchmark algorithm

The algorithm chosen for comparison was NSGA-II [9] which has been shown to be one of the better multi-objective Genetic Algorithms and has a number of high-quality, open-source software implementations freely available. Although various studies have been published which show competitive algorithms outperforming it on the family of ZDT problems [27], there are two main reasons for the choice of NSGA-II as the benchmark algorithm. First, the algorithm is very well known, due both to its inclusion in a number of free and commercial software packages and to its featuring in a range of published studies. Second, our ultimate goal in developing our TS algorithms is not absolute performance on a set of benchmark problems. Rather, our development is motivated by the requirements of real-world optimisation problems, for which we believe our TS algorithms to be well suited. Performance assessment on a set of well known problems against a well known algorithm then serves as validation of our approach and provides us with suggestions for algorithmic improvement.

Table 1
Test functions

Name	n_var	Objective functions	Variable bounds
ZDT1	30	$f_1(\bar{x}) = x_1$ $f_2(\bar{x}) = g(\bar{x})\left[1 - \sqrt{\frac{x_1}{g(\bar{x})}}\right]$ $g(\bar{x}) = 1 + 9\frac{\sum_{i=2}^{n\_var} x_i}{(n\_var-1)}$	$\bar{x} \in [0.0, 1.0]$
ZDT2	30	$f_1(\bar{x}) = x_1$ $f_2(\bar{x}) = g(\bar{x})\left[1 - (\frac{x_1}{g(\bar{x})})^2\right]$ $g(\bar{x}) = 1 + 9\frac{\sum_{i=2}^{n\_var} x_i}{(n\_var-1)}$	$\bar{x} \in [0.0, 1.0]$
ZDT3	30	$f_1(\bar{x}) = x_1$ $f_2(\bar{x}) = g(\bar{x})\left[1 - \sqrt{\frac{x_1}{g(\bar{x})}} - \frac{x_1}{g(\bar{x})}\sin(10\pi x_1)\right]$ $g(\bar{x}) = 1 + 9\frac{\sum_{i=2}^{n\_var} x_i}{(n\_var-1)}$	$\bar{x} \in [0.0, 1.0]$
ZDT4	10	$f_1(\bar{x}) = x_1$ $f_2(\bar{x}) = g(\bar{x})\left[1 - \sqrt{\frac{x_1}{g(\bar{x})}}\right]$ $g(\bar{x}) = 1 + 10(n\_var - 1) + s(\bar{s})$ $s(\bar{x}) = \sum_{i=2}^{n\_var}[x_i^2 - 10\cos(4\pi x_i)]$	$x_1 \in [0.0,1.0]$ $x_i \in [-5.0,5.0]$ $i = 2,\ldots,n\_var$
ZDT6	10	$f_1(\bar{x}) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$ $f_2(\bar{x}) = g(\bar{x})\left[1 - (\frac{f_1(\bar{x})}{g(\bar{x})})^2\right]$ $g(\bar{x}) = 1 + 9\left[\frac{\sum_{i=2}^{n\_var} x_i}{(n\_var-1)}\right]^{0.25}$	$\bar{x} \in [0.0, 1.0]$

5.3. Performance analysis

The performance assessment of multi-objective optimisation algorithms is more involved than the single-objective case. In single-objective optimisation, the output of an optimisation run is a single best objective function value obtained – thus performance can be directly compared on the basis of these values. In multi-objective optimisation, the output is set of solutions approximating the Pareto-optimal set, which we call an approximation set. The key to performing a performance assessment of a multi-objective optimisation algorithm is the ability to assign a measure of quality to an approximation set – in essence, reducing it to a single value – which then may be directly compared to other runs. There are a number of such performance indicators available, each indicator having different properties and measuring different aspects of quality of the approximation set. However, certain indicators are inconsistent with concepts of Pareto-dominance and, as such, are poor choices. Zitzler et al. [32] give a detailed discussion of the performance assessment of multi-objective optimisers. Following their advice, and that also given by Fonseca et al. [12], we use two performance indicators in this study: the unary epsilon indicator and the hypervolume indicator (both described below).

In this study, we have chosen not to include performance indicators that attempt to measure the degree to which solutions are evenly spread along the Pareto front. As discussed by Zitzler et al. [32], it is difficult to draw any meaningful conclusions from such indicators, except in cases where two sets of results are (or are very nearly) non-dominated by each other. Similarly, we have not included an indicator measuring distances of some kind to an ideal set. For certain problems, an ideal set cannot be calculated analytically. In addition, this kind of indicator does not offer any further insight into the relative performance of two algorithms than that provided by either the hypervolume or epsilon indicator. It only allows us to gauge absolute performance – as previously discussed, the nature of this study is largely comparative.

Further, as the optimisation algorithms under evaluation are not deterministic, the results from multiple runs must be collated and assessed. A suitable statistical test for determining whether certain algorithms tend to produce better approximation sets (using the performance indicators described) is the Kruskal–Wallis test [7]. This technique will also be described below.

5.3.1. Unary epsilon indicator

The unary epsilon indicator was proposed by Zitzler et al. [32] and makes direct use of the principle of Pareto-dominance, making it very intuitive. Consider two approximation sets A and B where A dominates B. The epsilon indicator is a measure of the smallest distance one would need to translate every point in B so that it dominates A. If set A is chosen to be a reference set, such that it dominates sets B and C, then B and C can be directly compared on the basis of the epsilon indicator with respect to the reference set. Thus, for a given reference set, the indicator is unary in nature. See Fig. 4 for a

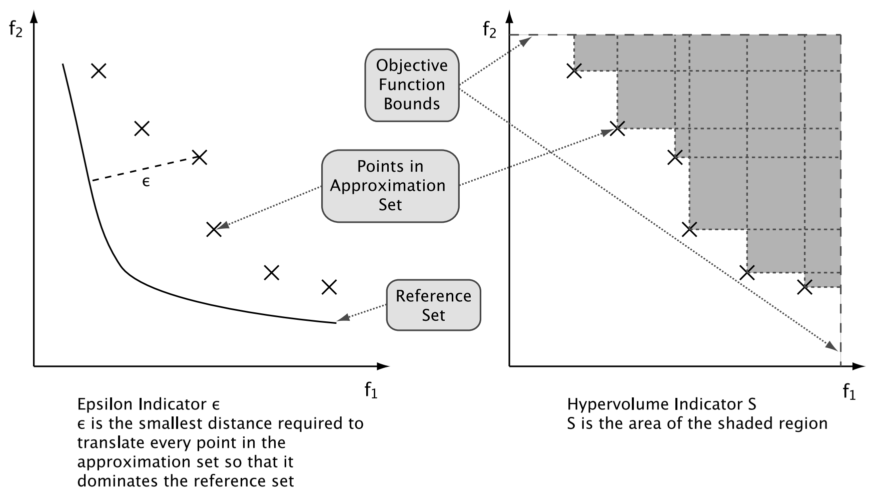
f₂
f₁
ε
Objective Function Bounds
Points in Approximation Set
Reference Set
Epsilon Indicator ε
ε is the smallest distance required to translate every point in the approximation set so that it dominates the reference set
f₂
f₁
Hypervolume Indicator S
S is the area of the shaded region

Fig. 4. Graphical representation of epsilon and hypervolume indicators.

graphical representation of this performance indicator.

5.3.2. Hypervolume indicator

The hypervolume indicator was proposed by Zitzler and Thiele [31] and its properties are discussed in depth in [32]. The two-dimensional case can be easily visualised and understood – each point in the approximation set forms a rectangle of given area with respect to a reference point that lies beyond the bounds of the approximation set. The hypervolume indicator is the area of the union of all these rectangles. This concept can easily be extended to higher dimensions. In effect, the hypervolume indicator measures the hyperspace enclosed by the approximation set. Fig. 4 also shows a graphical representation of this performance indicator.

5.3.3. Kruskal–Wallis test

The Kruskal–Wallis test is an extension of the Mann–Whitney test for two independent samples, allowing it to incorporate k independent samples, for $k \geqslant 2$. Using random samples of a quantity taken from k different populations, it is designed to determine whether certain populations tend to produce greater observed values of that quantity. Like the Mann–Whitney test, the Kruskal–Wallis test is a rank-based method – that is, all observations from the k populations are ranked together, and the ranks of individuals in a population are used to calculate the test statistic. Full details of this test (and the Mann–Whitney test) can be found in [7].

5.4. Experimental details

Each algorithm was run 45 times on each test function and the non-dominated sets obtained after 1000, 5000 and 10,000 function evaluations were archived and used to generate the performance measures. The random number generator for each of the 45 runs was initialised with a different seed, the seeds being constant across test functions and algorithms.

While previous studies [9,19] have assessed algorithm performance at up to 25,000 function evaluations, we felt a lower number of evaluations would yield results more applicable to real-world applications – Kipouros et al. performed aerodynamic shape optimisation with up to 3000 evaluations [24], while Knowles [25] has developed an optimiser to perform well on a maximum of 250 evaluations, in response to a real-world requirement.

The two TS algorithms were implemented in a C++ program and the parameter settings used are shown in Table 2. The TS algorithms were both initialised with a randomly selected point from the search space, selected differently according to the random number generator seed. The NSGA-II algorithm implementation used was that available in the PISA toolkit [3] using real-coded variables and simulated binary crossover (SBX) [9], and the parameter settings used are shown in Table 3.

The generation of the reference sets used for calculating the performance indicators and the calculation of the performance indicators themselves were also performed using software provided in the PISA toolkit. The Kruskal–Wallis tests and the generation of additional statistics were performed using the R statistical software environment [29].

The results were normalised such that all objective function values fell in the range 1.0–2.0. Thus, a reference point could be chosen as the nadir point of the entire set of results for a particular problem and evaluation count. This reference point was used in the hypervolume calculation. A reference set was

Table 2
Tabu Search parameter settings

Parameter	Description	Value
n_stm	Size of STM	20
n_regions	Divide search space into *n_var* * *n_regions* regions	2
intensify	Perform intensification when *i_local* = *intensify*	10
diversify	Perform diversification when *i_local* = *diversify*	20
restart	Reduce step sizes and restart when *i_local* = *restart*	50
SS	Initial step sizes (as % of variable range)	10%
SSRF	Step sizes are multiplied by this factor at *restart*	0.5
n_sample	Number of points randomly sampled (MOTS only)	6
select_interval	Perform variable selection every *select_interval* iterations (PRMOTS only)	20
n_selected	Number of active design variables (PRMOTS only)	$\frac{1}{2}n\_var$

Table 3
NSGA-II parameter settings

Parameter	Description	Value
n_pop	Population size	100
n_parent	Number of parents	100
n_child	Number of children	100
p_m	Mutation probability	0.1
p_r	Recombination probability	0.9
n_m	Mutation distribution index (for SBX operator)	15
n_r	Recombination distribution index (for SBX operator)	5

chosen as the 50% attainment surface of a random search and the indicator values were calculated relative to this reference set.

5.4.1. Run-time performance

Given the differences in complexity between MOTS/PRMOTS and NSGA-II, it is worth making a comment on the run-times of the algorithms under comparison. Detailed run-time figures were not calculated as the authors felt this was not worthwhile for two reasons. First, the execution time for all three algorithms under test was in the order of 30 seconds. For a real-world optimisation problem, it is not uncommon for the evaluation of a single-objective function value to take this amount of time; in many cases, objective function evaluations may take in the order of minutes or hours. Thus the proportion of the total computational cost attributable to the optimisation algorithm itself for most real-world applications is negligible.

Second, the performance of any code is highly implementation-specific. The implementation of NSGA-II used in this study was that provided by the PISA toolkit, which is written in C and makes heavy use of files as a means of inter-process communication. Our implementations of the TS algorithms were written in C++ and inter-process communication is achieved using the MPI protocol. There are clearly significant performance overheads associated with code-specific design choices, and this makes run-time comparison almost worthless.

A specific feature of our TS algorithms is the use of multiple memories, some of which are unbounded. The insertion of a point into the memories is, at worst, an n^2 operation and readers may rightly question the impact of this on performance. It is worth noting our comments above and our observations that the run-times of our TS algorithms were consistently smaller that those of the NSGA-II algorithm. The other potential performance impact of the use of memories in TS is the computer's RAM usage. Again, this is an implementation-specific detail. Through use of pointers, it is possible to minimise this RAM usage. Assume storage of 25,000 solutions comprising of 30 design variables and two objective functions: a total of 800,000 floating point numbers must be stored in the computer. Consider further, as a worst case scenario, that each of those 25,000 solutions is referenced by pointers in four separate memories: a further 100,000 pointer memory locations must be stored. A floating point number requires 4 bytes, as does a memory pointer. Thus, a total of $(800{,}000 + 100{,}000) \times 4$ bytes are required, or 3.6MB. This memory requirement, while making it unsuitable for embedded applications, is very low for modern desktop or server computer systems.

6. Results and discussion

The means and standard deviations of both performance indicators for all test conditions are shown in Table 4, and the results of the Kruskal–Wallis test are shown in Table 5. A lower number for both indicators signifies a "better" approximation set. As previously mentioned, the results for a particular problem and evaluation count were normalised and indicator values calculated relative to a reference set. This means that the results are relative and it is not possible to infer trends over different numbers of evaluations. For example, for problem ZDT3, both indicators have a greater value at 10,000 evaluations than at 5000 evaluations for all three algorithms – if the indicators were absolute this would suggest worse approximation sets at 10,000 evaluations than at 5000 evaluations, which is clearly wrong.

For the Kruskal–Wallis test, the null-hypothesis H0 was that no significant differences could be found in the distribution of results for the performance indicators at a confidence level $\alpha = 0.05$.

Table 4
Means and standard deviations of the hypervolume and epsilon performance indicators

Functions	Evaluations	Indicator	MOTS		PRMOTS		NSGA-II	
			Mean	SD	Mean	SD	Mean	SD
ZDT1	1000	hyp	0.2341	0.1623	0.1778	0.1396	0.1774	0.0201
		eps	0.2277	0.1748	0.1765	0.1524	0.1709	0.0190
	5000	hyp	0.0694	0.02170	0.0676	0.0490	0.0008	0.0002
		eps	0.0555	0.0162	0.0633	0.0467	0.0021	0.0003
	10,000	hyp	0.0425	0.0196	0.0331	0.0110	0.0002	0.0
		eps	0.0407	0.0170	0.0362	0.0113	0.0006	0.0
ZDT2	1000	hyp	0.1845	0.1413	0.1618	0.1151	0.2752	0.0362
		eps	0.1788	0.1492	0.1613	0.1305	0.3115	0.0371
	5000	hyp	0.0894	0.0344	0.0790	0.0185	0.0559	0.0315
		eps	0.1056	0.0372	0.0997	0.0312	0.1293	0.0724
	10,000	hyp	0.0500	0.0171	0.0461	0.0127	0.0559	0.0302
		eps	0.0725	0.0341	0.0795	0.0351	0.1294	0.0740
ZDT3	1000	hyp	0.1284	0.0590	0.1083	0.0250	0.3583	0.0172
		eps	0.1352	0.0637	0.1398	0.0396	0.3253	0.0282
	5000	hyp	0.0659	0.0283	0.0705	0.0275	0.1761	0.0012
		eps	0.1198	0.0537	0.1375	0.0537	0.1625	0.0016
	10,000	hyp	0.0834	0.0480	0.1018	0.0497	0.2995	0.0002
		eps	0.1938	0.0922	0.2302	0.0946	0.2701	0.0004
ZDT4	1000	hyp	0.4081	0.1006	0.4102	0.1025	0.0340	0.0155
		eps	0.3407	0.0830	0.3394	0.0958	0.0363	0.0127
	5000	hyp	0.1078	0.0862	0.0945	0.0881	0.0192	0.0100
		eps	0.1115	0.0795	0.0952	0.0834	0.0188	0.0087
	10,000	hyp	0.0104	0.0042	0.0103	0.0055	0.0269	0.0140
		eps	0.0113	0.0035	0.0121	0.0072	0.0277	0.0126
ZDT6	1000	hyp	0.4247	0.1289	0.3511	0.1410	0.2715	0.0723
		eps	0.2989	0.1235	0.2506	0.1272	0.2885	0.0702
	5000	hyp	0.3748	0.0756	0.3526	0.0757	0.0126	0.0064
		eps	0.2791	0.0725	0.2599	0.0815	0.0199	0.0065
	10,000	hyp	0.3176	0.0461	0.3201	0.0494	0.0008	0.0002
		eps	0.2319	0.0678	0.2379	0.0736	0.0020	0.0003

The p-values shown in Table 5 (e.g., $p(A > B)$) are the probabilities that an algorithm A exhibits better performance than algorithm B.

The testing procedure gave rise to 30 possible scenarios (5 test functions, 3 function evaluation levels and 2 performance indicators). In these 30 scenarios, the null-hypothesis was accepted in three instances – that is, it was not possible to discern any meaningful difference between the performance of the three algorithms in 10% of the scenarios.

The results in Table 5 allow us to compare the performance of the two TS variants, as well as that of the two variants plus NSGA-II. Considering solely the two TS variants, PRMOTS was the better algorithm in 18 (60%) of the scenarios whereas MOTS was the better algorithm in just 9 (30%) of the scenarios.

Considering all three algorithms, we find that NSGA-II was the better algorithm in 14 (47%) of the scenarios, whereas both MOTS *and* PRMOTS were better in 13 (43%) of the scenarios.

It is interesting to note that the indicators sometimes give contradictory results. For example, in Table 5 for problem ZDT4 at 10,000 evaluations, PRMOTS is found to be better than MOTS considering the hypervolume indicator, while the converse is true if the epsilon indicator is considered. While the indicators may be consistent with regard to the Pareto-dominance concept (as discussed in [32]), this does not necessarily mean that they are consistent among themselves. This highlights the problem of relying on just one indicator to assess an optimisation algorithm's performance.

Box-plots of the epsilon indicator on all five test functions after 10,000 evaluations are given in Figs. 5–9. The median, upper and lower quartiles and the extreme points for the 45 runs are shown in the box-plots. The results for the hypervolume indicator,

Table 5
Kruskal–Wallis p-values

Functions	Evaluations	Indicator	$p(A > B)$	$p(A > C)$	$p(C > B)$	Indicated best TS algorithm	Indicated best overall algorithm
ZDT1	1000	hyp	H0 hypothesis satisfied				
		eps	H0 hypothesis satisfied				
	5000	hyp	0.0	0.0006	0.0	PRMOTS	NSGA-II
		eps	0.0	0.1167	0.0	PRMOTS	NSGA-II
	10,000	hyp	0.0	0.00078	0.0	PRMOTS	NSGA-II
		eps	0.0	0.1486	0.0	PRMOTS	NSGA-II
ZDT2	1000	hyp	1.0	0.1667	1.0	PRMOTS	PRMOTS
		eps	1.0	0.2009	1.0	PRMOTS	PRMOTS
	5000	hyp	0.0	0.0161	0.0065	PRMOTS	NSGA-II
		eps	0.9997	0.2502	0.9999	PRMOTS	PRMOTS
	10,000	hyp	0.9980	0.1838	0.9999	PRMOTS	PRMOTS
		eps	0.9999	0.7529	0.9998	MOTS	MOTS
ZDT3	1000	hyp	1.0	0.0123	1.0	PRMOTS	PRMOTS
		eps	1.0	0.7119	1.0	MOTS	MOTS
	5000	hyp	1.0	0.8557	1.0	MOTS	MOTS
		eps	0.9998	0.9689	0.9665	MOTS	MOTS
	10,000	hyp	1.0	0.9496	1.0	MOTS	MOTS
		eps	H0 hypothesis satisfied				
ZDT4	1000	hyp	0.0	0.6287	0.0	MOTS	NSGA-II
		eps	0.0	0.5018	0.0	MOTS	NSGA-II
	5000	hyp	0.0	0.2197	0.0	PRMOTS	NSGA-II
		eps	0.0	0.0527	0.0	PRMOTS	NSGA-II
	10,000	hyp	1.0	0.3739	1.0	PRMOTS	PRMOTS
		eps	1.0	0.6400	1.0	MOTS	MOTS
ZDT6	1000	hyp	0.0	0.0	0.0012	PRMOTS	NSGA-II
		eps	0.6985	0.0015	0.9997	PRMOTS	PRMOTS
	5000	hyp	0.0	0.0003	0.0	PRMOTS	NSGA-II
		eps	0.0	0.0002	0.0	PRMOTS	NSGA-II
	10,000	hyp	0.0	0.3609	0.0	PRMOTS	NSGA-II
		eps	0.0	0.6460	0.0	MOTS	NSGA-II

Algorithm A = MOTS, B = NSGA-II, C = PRMOTS.

and the results after 1000 and 5000 evaluations are similar.

It is clear from the plots (and from Table 4) that NSGA-II in general exhibits less variation in performance than either TS variant, with the exception of its performance on ZDT4. This behaviour may be desirable in certain situations (e.g. [25]). Interestingly, results presented in [19] showed that NSGA-II outperformed MOTS on ZDT4 on optimisation runs of 25,000 evaluations, because MOTS becomes trapped in a local optimum. This suggests that NSGA-II has a greater ability to locate other areas of the design space; GAs in general have a larger stochastic element to them, whereas TS methods have at their core a local search, so this is to be expected.

Similarly (from Figs. 5–9 and Table 4), PRMOTS in general exhibits less performance variation than MOTS, although the distribution in results is broadly comparable, as one might expect given that the same basic algorithm is common to both variants.

There is only test function on which MOTS comprehensively outperforms PRMOTS is ZDT3, and it is instructive to analyse this closer. An attainment surface plot for this problem is shown in Fig. 10. An attainment surface plot is a way of visualising the output of n runs of an optimiser and shows the minimum surface in objective space achieved by $m < n$ of those runs. The technique is described in detail in [11] and software included with the PISA toolkit [3] was used to generate the plot. Fig. 10 shows the 18th or 60% attainment surface for both MOTS and PRMOTS on ZDT3 after 10,000 evaluations – this attainment surface

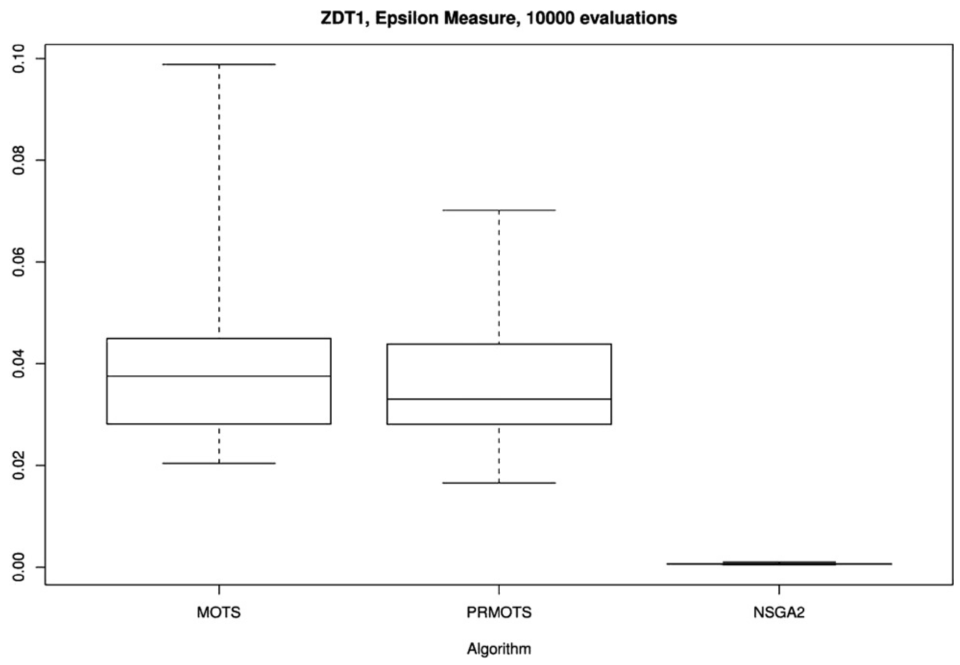

Fig. 5. Box-plot of epsilon indicator results for ZDT1 at 10,000 evaluations.

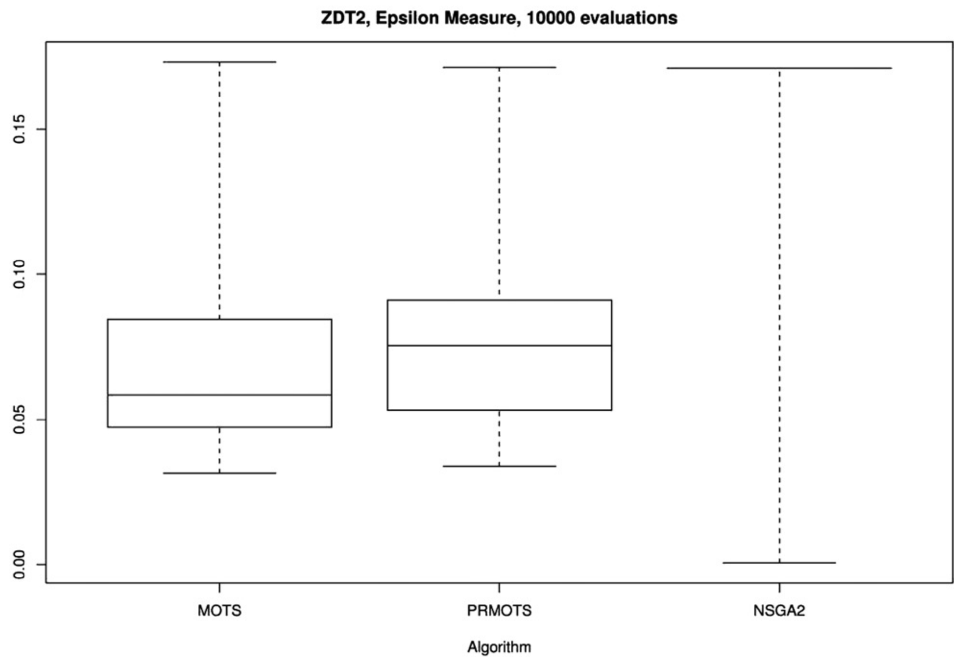

Fig. 6. Box-plot of epsilon indicator results for ZDT2 at 10,000 evaluations.

is presented because it clearly illustrates a difficulty that PRMOTS has on this problem. This test function has a disjointed Pareto front and therefore presents problems for optimisation algorithms.

It is immediately apparent from Fig. 10 that the reason that PRMOTS has performed poorly relative to MOTS on this problem is that it has failed to adequately resolve two sections of the Pareto front,

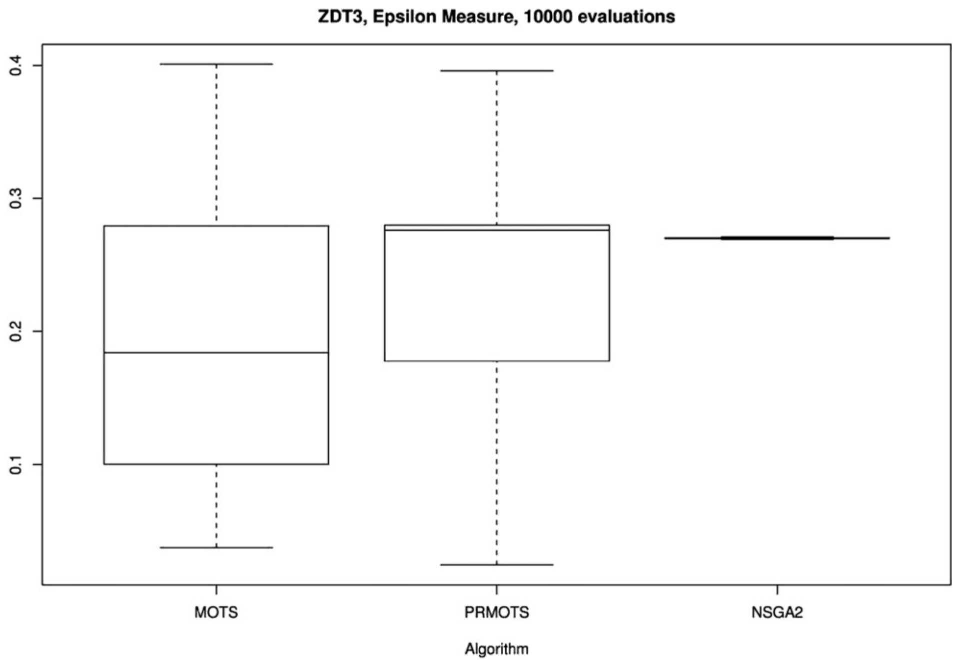

Fig. 7. Box-plot of epsilon indicator results for ZDT3 at 10,000 evaluations.

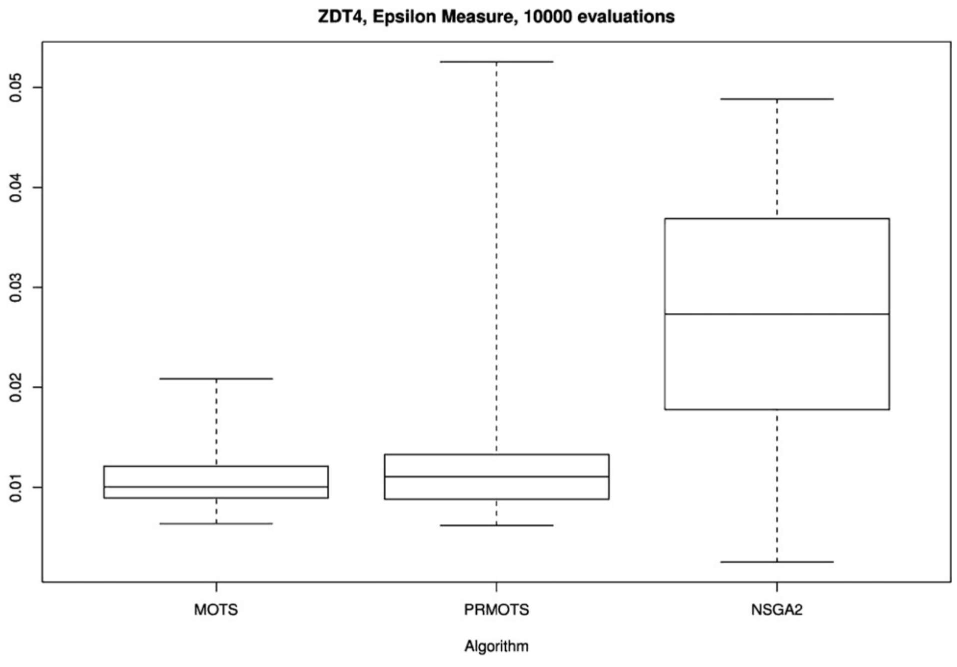

Fig. 8. Box-plot of epsilon indicator results for ZDT4 at 10,000 evaluations.

corresponding to $f_1 > 0.4$. Given that, on this problem, $x_1 = f_1$, it appears that PRMOTS has not fully explored the entire range of design variable x_1. One possible reason for this is that the variable selection scheme has "locked-in" x_1 to the lower end of its range; the optimisation algorithm may have quickly located the other two sections of the Pareto-front, thus the reference set used in the variable selection

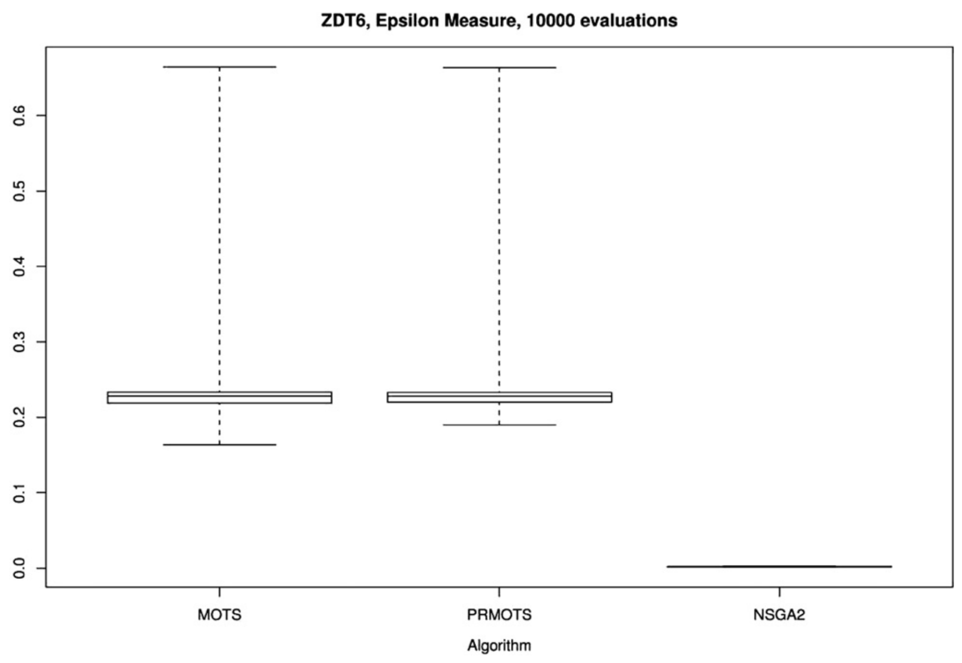

Fig. 9. Box-plot of epsilon indicator results for ZDT6 at 10,000 evaluations.

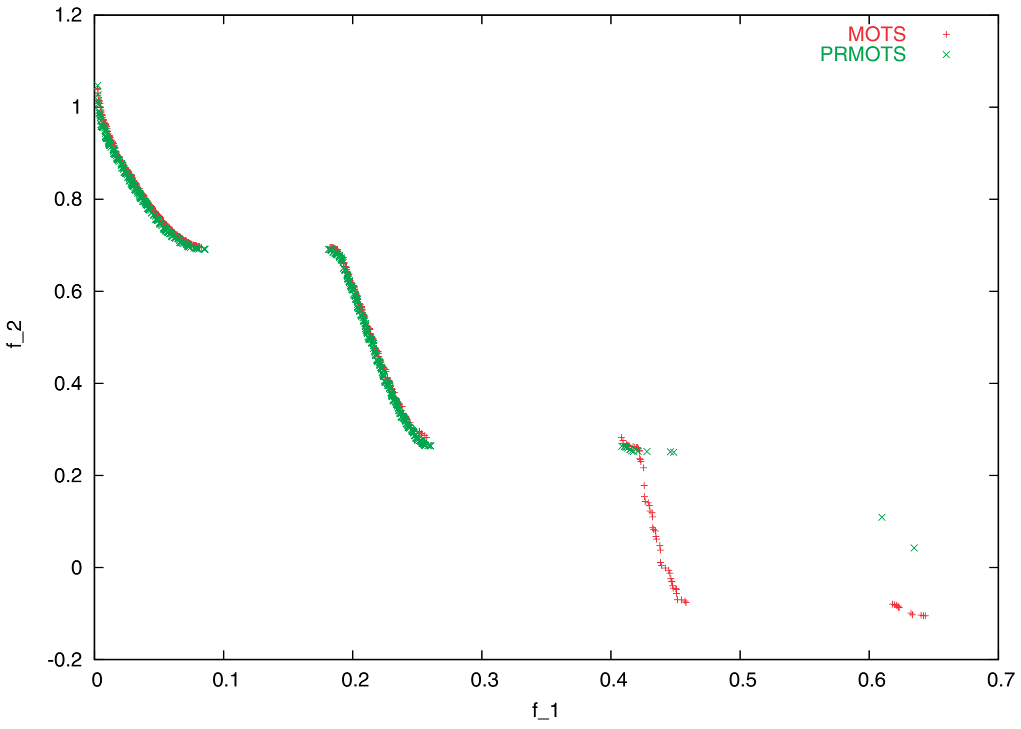

Fig. 10. 60% attainment surface plot for ZDT3, 10,000 evaluations, MOTS and PRMOTS.

scheme may not have shown enough diversity in x_1 to prevent x_1 from remaining dormant for much of the search. One possible, and simple, fix for this problem would be to introduce a stochastic element into the variable selection scheme, perhaps by means of roulette wheel selection of active design variables.

7. Conclusions

In this paper, we have presented two TS algorithms for use on continuous multi-objective optimisation problems. Both algorithms feature mechanisms to improve the efficiency of the local search component of TS. The first algorithm, MOTS, uses an element of random sampling to reduce the computational cost. A more advanced version of this algorithm, PRMOTS, uses a novel design variable selection scheme inspired by path relinking strategies common in discrete optimisation problems. This scheme effectively performs a sensitivity analysis of the design variables, while avoiding the computational cost of a full parametric analysis. This allows the dimensionality of the problem to be temporarily reduced, leading to greater local-search efficiency.

These two algorithms were tested on a set of five standard test functions and the results were compared with those from a leading multi-objective GA, NSGA-II. The results showed that the variable selection scheme gave a clear improvement in performance over the basic MOTS algorithm. Additionally, both algorithms performed comparably with NSGA-II. Overall, both TS algorithms exhibited greater performance variability than NSGA-II, but this was compensated by a greater likely improvement in the objective functions.

Based on these findings, it appears that our PRMOTS algorithm is competitive with NSGA-II and both could be expected to perform well on other multi-objective optimisation problems. NSGA-II may be a better choice where low performance variability is desired. However, as we have argued in this paper and as reported by other researchers [5], the local-search component and the constraint handling flexibility of TS makes it particularly attractive for problems which may be highly constrained, and we believe that PRMOTS is a good choice for these types of problems. A major shortcoming in the assessment of optimisation algorithms for use on real-world problems is the lack of a suitable set of benchmark problems, which accurately capture the main features of the problems of interest. Until such a set is developed, true performance comparison between algorithms is difficult and one must rely on inference from a less suitable set of benchmark problems.

7.1. Future work

There are a number of further algorithmic developments suggested by these results. The introduction of a random element in the parameter selection scheme would potentially alleviate some of the problems encountered by PRMOTS on ZDT3. Further, although the introduction of the parameter selection scheme is an improvement on the basic Hooke & Jeeves local search algorithm, there is still considerable scope for further enhancements in this area. For example, other than simple caching of solutions, the local search does not make use of the search memories (already present in the TS algorithm) in any intelligent manner: surrogate modelling, similar to that employed by the ParEGO algorithm [25] would offer such an opportunity. Finally, the parallelisation strategy employed places rather hard limits on the scalability of the algorithm across many processors, especially for problems with a low number of design variables. An algorithm employing multiple local searches with shared global memories – a multi-threaded TS – would be far more flexible in this regard, and may also offer raw algorithmic performance improvements over and above the parallel computational speed-up.

Acknowledgements

This research is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number GR/R64100/01. The authors would also like to thank Prof. Bill Dawes for his support and encouragement, and the anonymous reviewers for their helpful comments on the first version of this paper.

References

[1] R.M. Aiex, S. Binato, M.G.C. Resende, Parallel GRASP with path-relinking for job shop scheduling, Parallel Computing 29 (4) (2003) 393–430.

[2] A. Baykasoglu, S. Owen, N. Gindy, A taboo search based approach to find the Pareto optimal set in multiple objective optimization, Engineering Optimization 31 (1999) 731–748.

[3] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA – a platform and programming language independent interface for search algorithms, in: C.M. Fonseca, P.J. Fleming, E. Zitzler, K. Deb, L. Thiele (Eds.), Evolutionary Multi-Criterion Optimization (EMO 2003), LNCS, vol. 2632/2003, Springer-Verlag, Heidelberg, 2003.

[4] R. Caballero, X. Gandibleux, J. Molina, MOAMP – a generic multi-objective metaheuristic using an adaptive memory, Technical report, University of Valenciennes, 2004.

[5] L. Cavin, U. Fischer, F. Glover, K. Hungerbuehler, Multi-objective process design in multi-purpose batch plants using a Tabu Search optimization algorithm, Computers and Chemical Engineering 28 (4) (2004) 393–430.

[6] A.M. Connor, D.G. Tilley, A Tabu Search method for the optimisation of fluid power circuits, IMechE Journal of Systems and Control 212 (1998) 373–381.

[7] W.J. Conover, Practical Nonparametric Statistics, third ed., John Wiley, New York, 1999.

[8] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, John Wiley, Chichester, 2001.

[9] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[10] R. Duvigneau, M. Visonneau, Hybrid genetic algorithms and artificial neural networks for complex design optimization in CFD, International Journal for Numerical Methods in Fluids 44 (2004) 1257–1278.

[11] C.M. Fonseca, P.J. Fleming, On the performance assessment and comparison of stochastic multiobjective optimizers, in: H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN IV, LNCS, vol. 1141/1996, Springer-Verlag, Berlin, 1996.

[12] C.M. Fonseca, J. Knowles, L. Thiele, E. Zitzler, A tutorial on the performance assessment of stochastic multiobjective optimizers, An invited talk presented by J. Knowles at EMO 2005, Guanajuato, Mexico, 2005.

[13] A. Gaiddon, J.N. Greard, D. Pagan, D.D. Knight, Automated optimization of supersonic missile performance taking into account design uncertainties, Technical Report AIAA-2003-3879, AIAA, 2003.

[14] F. Glover, Scatter search and path relinking, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, New York, 1999.

[15] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, MA, 1997.

[16] M.P. Hansen, Tabu search for multiobjective optimization: MOTS, in: MCDM, Cape Town, South Africa, 1997.

[17] S.A. Harvey, The design optimisation of turbomachinery blade rows, Ph.D. thesis, Cambridge University, Engineering Department, 2002.

[18] R. Hooke, T.A. Jeeves, Direct search solution of numerical and statistical problems, Journal of the ACM 8 (1961) 212–229.

[19] D.M. Jaeggi, C. Asselin-Miller, G.T. Parks, T. Kipouros, T. Bell, P.J. Clarkson, Multi-objective parallel Tabu Search, in: X. Yao, E. Burke, J.-A. Lozano, J. Smith, J. Merelo-Guervos, J. Bullinaria, J. Rowe, P. Tino, A. Kaban, H-P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN VIII, LNCS, vol. 3242/2004, Springer-Verlag, Berlin, 2004.

[20] D.M. Jaeggi, G.T. Parks, T. Kipouros, P.J. Clarkson, A multi-objective tabu search algorithm for constrained optimisation problems, in: C.A. Coello Coello, A. Hernández Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization (EMO 2005), LNCS, vol. 3410/2005, Springer-Verlag, Berlin, 2005.

[21] A. Jameson, A perspective on computational algorithms for aerodynamic analysis and design, Progress in Aerospace Sciences 37 (2001) 197–243.

[22] D.F. Jones, S.K. Mirrazavi, M. Tamiz, Multi-objective meta-heuristics: An overview of the current state-of-the-art, European Journal of Operational Research 137 (2002) 1–9.

[23] W.P. Kellar, Geometry modelling in computational fluid dynamics and design optimisation. Ph.D. thesis, Cambridge University, Engineering Department, 2002.

[24] T. Kipouros, D.M. Jaeggi, W.N. Dawes, G.T. Parks, A.M. Savill, Multi-objective optimisation of turbomachinery blades using tabu search, in: C.A. Coello Coello, A. Hernández Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization (EMO 2005), LNCS, vol. 3410/2005, Springer-Verlag, Berlin, 2005.

[25] J.D. Knowles, ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multi-objective optimization problems, IEEE Transactions on Evolutionary Computation 10 (1) (2006) 50–66.

[26] J.D. Knowles, D.W. Corne, Approximating the nondominated front using the Pareto archived evolution strategy, Evolutionary Computation 7 (3) (1999) 1–26.

[27] A.J. Nebro, F. Luna, E. Alba, New ideas in applying scatter search to multiobjective optimization, in: C.A. Coello Coello, A. Hernández Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization (EMO 2005), LNCS, vol. 3410/2005, Springer-Verlag, Berlin, 2005.

[28] T. Okabe, Y. Jin, M. Olhofer, B. Sendhoff, On test functions for evolutionary multi-objective optimization, in: X. Yao, E. Burke, J.-A. Lozano, J. Smith, J. Merelo-Guervos, J. Bullinaria, J. Rowe, P. Tino, A. Kaban, H-P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN VIII, LNCS, vol. 3242/2004, Springer-Verlag, Berlin, 2004.

[29] R Development Core Team, R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, 2004, ISBN 3-900051-07-0.

[30] A. Suppapitnarm, K.A. Seffen, G.T. Parks, P.J. Clarkson, A simulated annealing algorithm for multiobjective optimization, Engineering Optimization 33 (2000) 59–85.

[31] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms – a comparative case study, in: W. Ebeling, I. Rechenberg, H.-P. Schwefel, H.-M. Voigt (Eds.), Parallel Problem Solving from Nature – PPSN V, LNCS, vol. 2632/1998, Springer-Verlag, Heidelberg, 1998.

[32] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, V. Grunert da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 117–132.