Monte Carlo Tree Search with Macro-Actions and Heuristic Route Planning for the Multiobjective Physical Travelling Salesman Problem

Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling
Department of Computer Science
York Centre for Complex Systems Analysis
University of York, UK
Email: edward.powley@york.ac.uk, dw830@york.ac.uk, peter.cowling@york.ac.uk

***Abstract*—This paper describes our entry to the *Multiobjective Physical Travelling Salesman Problem (MO-PTSP)* competition at the IEEE CIG 2013 conference. MO-PTSP combines the classical Travelling Salesman Problem with the task of steering a simulated spaceship on the 2-D plane, requiring that the controller minimises the three objectives of time taken, fuel consumed and damage incurred.**

Our entry to the MO-PTSP competition builds upon our winning entry to the previous (single-objective) PTSP competitions. This controller consists of two key components: a pre-planning stage using a classical TSP solver with a path cost measure that takes the physics of the problem into account, and a steering controller using *Monte Carlo Tree Search (MCTS)* with macro-actions (repeated actions), depth limiting and a heuristic fitness function for nonterminal states. We demonstrate that by modifying the two fitness functions we can produce effective behaviour in MO-PTSP without the need for major modifications to the overall architecture.

The fitness functions used by our controller have several parameters, which must be set to ensure the best performance. Given the number of parameters and the difficulty of optimising a controller to satisfy multiple objectives in a search space which is many orders of magnitude larger than that encountered in a turn-based game such as Go, we show that informed hand tuning of parameters is insufficient for this task. We present an automatic parameter tuning method using the *Covariance Matrix Adaptation Evolution Strategy (CMA-ES)* algorithm, which produced parameter settings that dominate our hand tuned parameters. Additionally we show that the robustness of the controller using hand tuned parameters can be improved by detecting when the controller is trapped in a poor quality local optimum and escaping by switching to an alternate fitness function.

I. Introduction

Monte Carlo Tree Search (MCTS) is a game tree search algorithm that has enjoyed recent success and attention in many domains [1]. Most notable amongst these domains is the ancient Chinese board game Go [2], in which heuristically guided minimax search is weak but MCTS-based players approach the level of top professional human players. MCTS has several useful properties which make it suitable for developing game AI. MCTS is *aheuristic*: in its most basic form, it requires no game-specific knowledge other than a forward simulation model. MCTS is *anytime*: it can be halted after any amount of computational time has elapsed and will give a reasonable result, although more time leads to better decisions. MCTS exhibits *asymmetric* tree growth: instead of searching the tree to a fixed depth, it focuses its attention on the most promising regions of the tree (but potentially explores the entire tree, given enough time). MCTS works by executing a large number of *playouts* from the current state to some point in the future (e.g. the end of the game, or after a fixed number of moves). Initially the playouts are random, but each playout results in a node being added to a partial search tree and a reward signal being backpropagated through the tree. The tree informs the policy used by subsequent playouts, so a strong policy is reinforced.

The *Physical Travelling Salesman Problem (PTSP)* [3] extends the classical Travelling Salesman Problem (TSP) [4] from a discrete optimisation problem to a discretised real-time planning and control problem. In the TSP, moving from one node to another is an atomic action incurring a known fixed cost. In the PTSP, moving from one node to another involves steering a simulated physical object on the 2D plane, and so the cost depends not only on the distance between the nodes but also on the path that is steered.

PTSP competitions were held at the WCCI 2012 and CIG 2012 conferences [5], [6] and in both competitions our controller, PUROFVIO, proved to be significantly stronger than the other AI players and on a par with the strongest human players [7]. Our controller uses MCTS for steering, using *macro-actions* (repeated actions) to reduce the granularity of the action space and hence the depth of the tree as well as to increase the time available for each decision, and depth limiting with a heuristic fitness function to reduce the length of playouts. The anytime nature of MCTS was leveraged to ensure the AI chooses actions within the allocated time of 40ms per decision.

In the PTSP, the aim is to minimise the time taken to collect all waypoints on the map. In this paper we describe our entry to the *Multiobjective Physical Travelling Salesman Problem (MO-PTSP)* competition held at the CIG 2013 conference [6]. MO-PTSP introduces two further objectives into the PTSP: applying thrust to the ship causes *fuel* to be spent (although passing through waypoints or collecting optional fuel tanks allows the ship to regain fuel) and colliding with certain walls or driving through lava causes *damage* to the ship. A successful entry to the competition

978-1-4673-5311-5/13/$31.00 ©2013 IEEE

must seek to minimise all three of these incommensurable objectives simultaneously.

This paper describes the modifications we made to PUROFVIO for the MO-PTSP. The resulting controller is named PUROFMOVIO. The architecture of our controller is as described in [7], [3], consisting of a *route planner* that selects the waypoint order using a TSP heuristic (with a cost function that takes into account the physics of the PTSP), and a *steering controller* that pilots the ship between waypoints using MCTS (with macro-actions, depth limiting and heuristic evaluation). The heuristic evaluations used by PUROFMOVIO are more complex than those in PUROFVIO, to take account of the new game elements and objectives.

Since we introduced a substantial number of parameters to the controller, we elected to conduct automatic parameter tuning for our controller. This was necessary for two reasons: some of the new parameters may interact with each other in ways that are difficult to predict, and the multi-objective nature of the problem introduces compromises into the optimisation process (for example optimising for damage may increase time spent and fuel used). We employ the *Covariance Matrix Adaptation Evolution Strategy (CMA-ES)* algorithm [8] for parameter tuning, but use non-dominated sorting of individuals [9] to handle the multiple objectives. Automatic parameter tuning successfully produced controllers which perform better than our hand tuned controllers.

We also show how the reliability of our hand tuned controller can be improved by introducing a mechanism to detect when the player is "stuck" in a poor quality local optimum and switching to an alternative *panic mode* fitness function designed to escape these situations. Other authors have investigated the idea of dynamically modifying the fitness function to allow local search to escape poor quality local optima, such as the variable fitness function approach of Remde *et al* [10] and the guided local search approach of Tsang and Voudouris [11]. In our approach the "local search" algorithm is depth-limited MCTS, and the movement of the ship around the map during the game itself plays the role of local search's exploration of the space. We detect local optima based on the statistics collected in the MCTS tree and respond by switching to a hand-designed set of weights designed to move to a new, higher fitness region by aggressively seeking the next node at any cost.

This paper is structured as follows. Section II describes the Multiobjective Physical Travelling Salesman Problem. Section III describes our MO-PTSP controller PUROFMOVIO, highlighting how it differs from our PUROFVIO controller for PTSP. In particular, Section III-D describes the panic mode enhancement for escaping poor quality local optima. Section IV describes how we automatically tuned the various algorithm and fitness function parameters, and investigates the performance of our controller with both hand tuned and automatically tuned parameters. Finally Section V gives some concluding remarks and directions for future work.

II. MULTIOBJECTIVE PTSP

This section gives a description of the MO-PTSP, outlining the differences between it and the PTSP.

A. PTSP

The *Physical Travelling Salesman Problem (PTSP)* is a real-time game played on a 2-dimensional *map*. The map has *walls* as obstacles, as well as a number of *waypoints*. In the MO-PTSP competition as in the WCCI 2012 PTSP competition, each map has 10 waypoints; maps in the CIG 2012 PTSP competition had 30–50 waypoints. The map itself is represented as a bitmap, where each pixel is either a wall or empty space. The game proceeds in discrete time steps, nominally one every $40\,\mathrm{ms}$ (i.e. 25 per second). The player controls a *spaceship*, similar to the one in the classic video game *Asteroids* [12]: the ship can rotate at a constant angular speed and can apply thrust in the direction that it is currently pointing. The combinations of rotation (clockwise, anticlockwise or none) and thrust (on or off) give a total of $3 \times 2 = 6$ actions, one of which must be chosen in each time step. The ship is subject to simulated Newtonian physics: thrust applies a force in the direction the ship is pointing, but if the ship already has momentum then this may not be the direction of travel. Collisions with walls do not damage the ship, and in physical terms are inelastic: the ship bounces away from the wall with reduced momentum.

The aim of the game is to pilot the ship through all waypoints on the map in as few time steps as possible. As the map is potentially not seen in advance, the AI controller is given 1 second of preprocessing time before the game begins. The controller must then specify an action every $40\,\mathrm{ms}$. The player is disqualified if a certain number of time steps elapse (800 in this competition) without the ship having collected a waypoint; otherwise the game ends once all waypoints have been collected.

B. MO-PTSP

The *Multiobjective Physical Travelling Salesman Problem (MO-PTSP)* modifies the PTSP in several ways. The key difference is the addition of two counters, for *fuel* and *damage*. Both counters start at 5000. The fuel counter is decreased by 1 on every time step in which the controller applies thrust, and increased by 50 (up to a maximum of 5000) when a waypoint is collected. The damage counter is decreased by 10 when the ship collides with a wall. New map elements, described below, also have effects on the two counters. The aim of the game is to collect all waypoints while minimising the three objectives of time steps taken, fuel consumed and damage incurred. If the fuel counter reaches zero, the ship is rendered unable to thrust. If the damage counter reaches zero, the ship is destroyed and the game ends. However in practice we have found that there is virtually no danger of these conditions occurring in normal play.

Maps for MO-PTSP contain several new elements in addition to walls and waypoints:

- *Red walls*, collisions with which inflict 30 damage points to the ship and have a lower coefficient of restitution (i.e. remove more of the ship's momentum) than normal walls.
- *Blue walls*, collisions with which inflict no damage and have a higher coefficient of restitution than normal walls. Colliding with a suitably placed blue wall can

be an effective way of changing the direction of travel, using less time and less fuel than rotating the ship and thrusting.
- Areas of *lava*. The ship incurs 1 point of damage for every time step it spends in lava. Most of the maps are designed such that lava cannot be avoided completely (e.g. with waypoints located in areas of lava), so the controller must instead aim to spend as little time in the lava as possible.
- *Fuel tanks*. Collecting a fuel tank increases the ship's fuel counter by 250, up to a maximum of 5000.

C. Competition scoring

The AI-versus-AI competition is played across 20 unseen maps. For each map, a competitor receives 1 point if it is not Pareto dominated by any other competitor; that is, if there is no competitor that completed the map with equal or better scores for all three of time, fuel and damage, and strictly better for at least one of these objectives. On the Pareto front, a competitor is awarded an additional 2, 9 or 14 points if it outperforms all other competitors on one, two or three of the objectives respectively. Thus seeking to optimise two of the objectives at the expense of the third is a viable strategy, but seeking to optimise only one is less likely to perform well.

The AI-versus-human competition is played across the 10 seen maps provided with the competition software framework. The scoring system is the same as for the AI-versus-AI competition. Human competitors play the game in real-time via a Java applet on the competition website [6].

In the PTSP competitions, competitors were scored on the basis of the number of waypoints they collected and the time taken to do so. The MO-PTSP competition is stricter with regards to number of waypoints collected: a competitor failing to collect all waypoints on a map receives no points for that map.

III. Design of the controller

The architecture of PUROFVIO for PTSP is described fully in [7], [3], and the architecture of PUROFMOVIO for MO-PTSP is broadly similar. The controller has three main components:

1) The *distance mapper*, which precomputes the shortest-path distance between every waypoint and every other non-wall pixel on the map using a modified version of the scanline flood fill algorithm [13].
2) The *route planner*, which uses the multiple fragment heuristic [14] and 3-opt local search with first improvement [15], [16] for the classical TSP to plan the order in which to visit the waypoints. The edge weights in this TSP instance are the shortest-path distances between waypoints, queried from the distance maps. Our 3-opt implementation uses a custom cost function, described below, to account for the physics of the problem.
3) The *steering controller*, which is responsible for specifying, every 40 ms, an action for the ship to execute. It uses the UCT variant of MCTS [17], with *macro-actions* (actions repeated for a fixed number of consecutive time steps [7], [3]) and depth limiting (cutting off the playout, which includes both tree descent and simulation, after a fixed number of moves) to address the huge state space in PTSP. Nonterminal states are evaluated using a heuristic fitness function described below.

Fig. 1. An example of a distance map with a weighting of $\gamma = 1$, so that paths through lava have twice the weight of paths through empty space. The contour lines correspond to distances a multiple of 25 pixels from the origin (shown as a dot). Note that in regions of lava (shaded light grey) the contour lines are closer together.

The remainder of this section describes the evaluations used for MO-PTSP, and other differences from the PTSP controller entered into the 2012 competitions.

A. Including lava in distance maps

The inclusion of lava in the map effectively changes the edge weights for the route planner, as passing through lava is more costly than passing through space (both have the same time cost, but the former also has a damage cost). Lava has a similar effect on the best path to steer between two waypoints, as a short path through lava may be more costly than a slightly longer path without lava. We address both of these issues by factoring lava into the distance maps. For PTSP the distance mapper assumes a weight of 1 between horizontally or vertically adjacent pixels. For MO-PTSP we use the same distance when neither pixel contains lava; if one pixel contains lava then we use a weight of $1 + \frac{1}{2}\gamma$; if both pixels contain lava then we use a weight of $1 + \gamma$. Here γ is a constant to be tuned. For diagonally adjacent pixels, the distance mapper uses the above weights multiplied by $\sqrt{2}$. The effect of this modification on the distance maps is shown in Figure 1.

B. Route cost function

An efficient route for PTSP must take account of the physics of the problem. This is illustrated in Figure 2, where a TSP solver taking only edge lengths into account chooses a suboptimal route. We address this problem by using a heuristic cost function during 3-opt local search, in place

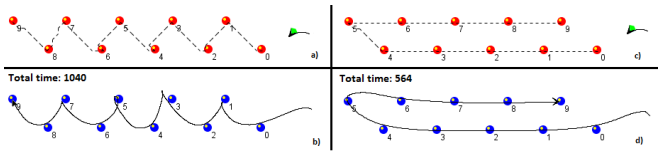

Fig. 2. Illustration of route choice in PTSP: despite being shorter in terms of edge lengths, the route on the left involves more sharp turns and therefore takes longer for the steering controller to follow. [3, Figure 3].

of the more traditional sum of edge weights. The cost for a route $\langle w_1, \ldots, w_n \rangle$ is

$$\begin{aligned} c(\langle w_1, \ldots, w_n \rangle) = &\sum_{i=1}^{n-1} d(w_i, w_{i+1})^{\beta_1} \\ &+ \beta_2 \sum_{i=1}^{n-1} \frac{d(w_i, w_{i+1})}{e(w_i, w_{i+1})} \\ &+ \beta_3 \sum_{i=1}^{n-2} \theta(w_i, w_{i+1}, w_{i+2}) \end{aligned} \quad (1)$$

where β_1, β_2 and β_3 are constants to be tuned. Here $d(w_i, w_{i+1})$ is the distance in pixels between waypoints w_i and w_{i+1}, queried from the distance map for w_i, $\frac{d(w_i, w_{i+1})}{e(w_i, w_{i+1})}$ is a measure of the *indirectness* of the edge (how much the shortest path deviates from a straight line between the waypoints), and $\theta(w_i, w_{i+1}, w_{i+2})$ is a measure of the *change in angle* between the edges on the route entering and leaving w_{i+1}. Full details of how indirectness and change in angle are computed are given in [7].

Note the nonlinear mapping applied to the edge weight $d(w_i, w_{i+1})$ in the first term: if $\beta_1 > 1$, routes containing very long edges are strongly penalised. (Typically $d(w_i, w_{i+1})$ is between 100 and 1000, so β_1 is tuned close to 1 to avoid making the effect too extreme.) This helps to avoid situations where the route planner suggests a route that cannot reasonably be followed without timing out. In PUROFVIO this nonlinearity was not included, i.e. $\beta_1 = 1$.

We found through testing that merely allowing the steering controller to opportunistically collect fuel tanks as it passes them was sometimes insufficient. Thus we include fuel tanks as optional waypoints in the graph constructed by the route planner. The inclusion of optional waypoints with rewards for collection in the standard TSP is known as *prize collecting* [18]. As planning a longer route to pick up fuel tanks may not always be worthwhile, we introduce a boolean parameter β_4 to control whether fuel tanks are included. If they are, we tackle the route planning problem by modifying the heuristic cost used by 3-opt for a candidate route $\langle v_0, v_1, \ldots, v_n \rangle$ where v_0 is the starting position and $v_1, \ldots, v_n$ are waypoint and fuel tank positions. Let c denote the route cost function without fuel tanks (Equation 1). Let w be the index of the last waypoint in $v_1, \ldots, v_n$ (so that $v_{w+1}, \ldots, v_n$ consists solely of fuel tanks) and let $f(w)$ be the number of fuel tanks in $v_1, \ldots, v_w$. Then the heuristic cost function for the candidate route is defined as

$$c'(\langle v_0, v_1, \ldots, v_n \rangle) = c(\langle v_0, v_1, \ldots, v_w \rangle) - \beta_5 f(w) \quad (2)$$

for a constant $\beta_5 \geq 0$. That is, the cost function only considers the portion of the route up to and including the last waypoint, and subtracts a bonus from the cost for each fuel tank included in this portion. This makes fuel tanks optional without modifying 3-opt to explicitly consider them as such: the decision not to collect a fuel tank is made by pushing the tank to the end of the route, after the last waypoint. The bonus is required since a route with fuel tanks almost certainly has a higher cost than one without, so otherwise the route planner would have no incentive to include them.

Collecting two fuel tanks in quick succession is rarely a good idea: collecting a fuel tank when the ship's fuel counter is nearly full is wasteful, and fuel tanks do not reset the counter for maximum time between waypoints. For similar reasons it is rarely wise to collect a fuel tank at the very beginning of the game. To discourage such routes, we increase the weights of edges between fuel tanks and between the starting position and a fuel tank by a constant β_6.

C. Fitness function for steering

The fitness function used by the steering controller to evaluate nonterminal states is a linear combination of seven terms:

$$\begin{aligned} v(s) = &\alpha_1 m(s) + \alpha_2 e(s) + \alpha_3 (1 - d(s)) \\ &+ \alpha_4 f_u(s) + \alpha_5 f_t(s) + \alpha_6 d_u(s) + \alpha_7 d_c(s) \,. \end{aligned} \quad (3)$$

Here $\alpha_1, \ldots, \alpha_7$ are constants to be tuned. The first three terms are designed to guide the steering controller towards the next waypoint, give a large reward for collecting the waypoint, and achieve a favourable position from which to set off towards the next waypoint. The $m(s)$ term measures the number of waypoints on the route collected so far, the $e(s)$ term counts the number of waypoints collected out of route order, and the $(1 - d(s))$ term measures the fraction of the distance travelled towards the next waypoint, according to the distance map. Whether the player is rewarded or penalised for early collection of waypoints depends on the sign of α_2. When choosing the parameter values it is important to set $\alpha_1 > \alpha_3$ so that there is a discontinuous jump in fitness upon collecting the next waypoint. This helps to ensure that states where the waypoint is collected are evaluated as strictly better than those where it is not. If this condition does not hold, the controller has a tendency to drive close to the next waypoint but avoid collecting it [7].

The terms $f_u(s)$ and $d_u(s)$ are the units of fuel consumed and damage incurred respectively. In general the coefficients α_4 and α_6 for these terms should be small, as making them too large can introduce poor quality local optima into the fitness landscape. Setting α_6 too high results in a controller that avoids lava at all costs, even when driving through lava is necessary to collect the next waypoint. Even worse, setting α_4 too high results in a controller that prefers to remain motionless at the starting point rather than spend any fuel.

The term $f_t(s)$ counts the number of fuel tanks collected out of route order, i.e. not already counted in $m(s)$. If fuel tanks are not included in the route, this term rewards the controller for opportunistically collecting fuel tanks whilst driving between waypoints. The term $d_c(s)$ measures the damage from wall collisions in the game up to state s. Collisions with normal walls increment the measure by 1, and collisions with red walls by 3 (as they do three times

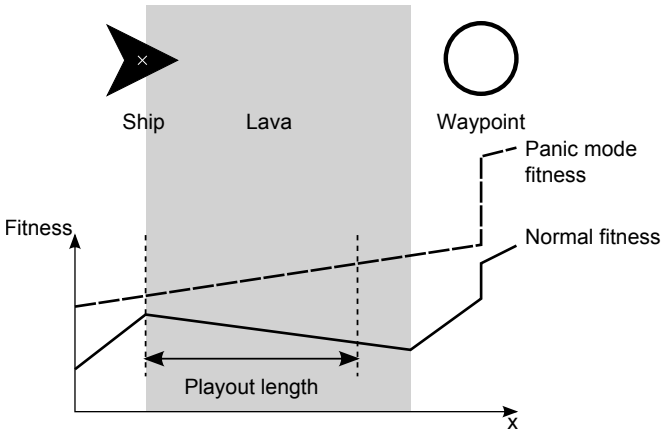

Fig. 3. Illustration of a low quality local optimum in a simplified 1-dimensional version of MO-PTSP. The fitness function gives a reward for approaching the next waypoint but a penalty for damage incurred by driving through lava (the ship takes damage only if its geometric centre, marked $\times$, is over the lava). The graph plots the fitness as a function of the x-axis position of the ship, assuming constant velocity. The playout length is insufficient for the search to determine that driving through the lava eventually leads to the next waypoint and thus a higher fitness. Thus the controller is stuck in the local optimum near the edge of the lava. The panic mode fitness function ignores damage from lava, so has no local optimum here.

the damage to the ship). Collisions with blue walls, which do not damage the ship, are not counted. One could argue that f_t and d_c are superfluous, as collecting fuel tanks and colliding with walls are already measured by f_u and d_u respectively. However including them as separate terms allows their weights to be tuned separately to produce behaviours not possible with f_u and d_u alone. For example we can set α_7 high to obtain a controller that avoids collisions, without the risk of setting α_6 too high and making the controller too reluctant to drive across lava.

D. Panic mode

Given the complexity of the fitness function used by steering, and the potential need to incur penalties in order to progress towards the next waypoint, the steering controller may sometimes become stuck in a low quality local optimum for the fitness function which does not correspond to reaching (or making progress towards) the next waypoint. This is illustrated in Figure 3. This type of problem arises when one local optimum is separated from a better one by a "valley" of low fitness, but the search is too short-sighted to see across the valley. This is similar to the *horizon effect* in minimax search [19], where the AI is unable to accurately evaluate a move due to some consequence of the move occurring beyond the depth cutoff. In the PTSP competitions we solved this type of problem by careful tuning of the parameters, but given the additional parameters present in our MO-PTSP controller, preliminary experiments highlighted a need to address the problem of local optima directly. We introduce *panic mode*, a mode of operation in which the controller's goal is to collect the next waypoint (and avoid losing the game by timing out) at any cost.

When the search for the next macro-action has used $\frac{3}{4}$ of its time budget, i.e. when the macro-action currently being executed is 75% complete, we compare the fitness of the current root state v_{current} with the average backpropagated reward v_{tree} at the root of the tree. If

$$v_{\text{tree}} + \alpha_0 \leq v_{\text{current}} \tag{4}$$

(where α_0 is a constant), we conclude that the search is failing to find lines of play that make progress towards the next waypoint. The value v_{tree} is a measure of the expected fitness at the end of the playout, so inequality (4) implies that the search has failed to find a line of play that improves upon the current state. In this case we abandon the current search, and spend the remaining time searching for the next macro-action using an alternative set of fitness function parameters. These parameters use a reduced search depth of 3 (the hand tuned search depth is 8), ignore fuel and damage ($\alpha_4 = \alpha_5 = \alpha_6 = \alpha_7 = 0$), and give a very high reward for collecting the next waypoint ($\alpha_1 = 10$, compared to the hand tuned value $\alpha_1 = 1$). These settings cause the controller to greedily collect the next waypoint ignoring other considerations.

IV. Tuning and performance

A. Parameter tuning

Our controller has a number of parameters and settings, listed in Table I. For the 2012 PTSP competitions, there was a small number of parameters and good values were chosen by hand. Given the increased number of parameters and the multiple objectives in the MO-PTSP, the effect of changing parameters is unpredictable and hand tuning is unreliable. Instead we opted to tune parameters automatically using a multi-objective optimisation approach. For some parameters we retained the hand tuned values in order to improve the effectiveness of parameter optimisation. For example, panic mode was implemented as a last resort measure, so the parameters governing when it is activated were not tuned during these experiments.

To perform optimisation of parameters we used the *Covariance Matrix Adaptation Evolution Strategy (CMA-ES)* algorithm [8]. The CMA-ES algorithm samples a new population of individuals at each iteration and uses the fitness of the individuals to adjust the distribution over parameters. An individual consists of an assignment of value to each parameter, and fitness was measured by testing these values on all 10 starter kit maps and averaging the time, fuel and damage scores as proportions of the maximum observed scores. The CMA-ES algorithm updates the distribution over parameters at each iteration using the ranking of individuals in each population and not their relative fitness values. When applying the CMA-ES algorithm to a multi-objective problem there is not a single fitness value to perform a ranking with. To handle the multiple objectives in ranking of individuals, we modified the CMA-ES algorithm in a similar way to [9] using non-dominated sorting on the population of parameter vectors generated on each CMA-ES iteration. Non-dominated sorting assigns all individuals in a population which are not dominated by any other member of the population rank 1, then assigns all unranked individuals non dominated by another unranked individual rank 2, and so on. To ensure that the algorithm chooses robust parameter values that collect all waypoints, we consider an individual that fails to collect all waypoints to be dominated by one that succeeds,

Parameter	Description	Type	Range	2012 PTSP	Hand tuned	CMA-ES Run 1	CMA-ES Run 2	CMA-ES Run 3
Distance mapper								
γ	Distance weight for lava	Real	$[0, 2]$	0	0.5	0.0573	0.493	0.732
Route planner								
β_1	Exponent for edge weight	Real	$[1, 2]$	1	1.5	1.76	1.38	1.11
β_2	Penalty for indirect paths between nodes	Real	$[0, 500]$	150	150	341	332	92.3
β_3	Penalty for sharp turns at nodes	Real	$[0, 500]$	80	80	-41.8	498	28.8
β_4	Include fuel tanks in the route?	Bool	$\{\text{false}, \text{true}\}$	false	true	true	false	true
β_5	Bonus for each fuel tank in route	Real	$[0, 500]$	–	200	416	–	516
β_6	Penalty for consecutive fuel tanks in route	Real	$[0, 1000]$	–	1000	240	–	728
Steering controller								
T	Macro-action length	Int	$\{5, \ldots, 20\}$	15	15	9	12	11
d	Playout depth	Int	$\{1, \ldots, 10\}$	8	8	7	7	7
C	UCB1 exploration constant	Real	$[0, 2]$	1.0	1.0	0.505	0.674	0.428
α_0	Score threshold for panic mode	Real	$[0, 1]$	–	0.1	0.1	0.1	0.1
Fitness function								
α_1	Weight for number of waypoints collected	Real	$[0, 1]$	1.0	1.0	1.70	1.32	0.807
α_2	Weight for waypoints collected out of order	Real	$[-1, 1]$	-1.0	-1.0	0.504	0.109	0.659
α_3	Weight for distance to next route node	Real	$[0, 1]$	0.75	0.75	1.61	1.24	0.720
α_4	Penalty per unit of fuel used	Real	$[0, 0.01]$	0	0.001	0.00179	0.00157	0.000309
α_5	Bonus per fuel tank collected	Real	$[0, 1]$	0	0.2	0.552	0.318	0.287
α_6	Penalty per unit of damage	Real	$[0, 0.01]$	0	0.002	0.00293	0.00118	0.00175
α_7	Penalty per wall collision	Real	$[0, 0.5]$	0	0.3	0.314	0.108	0.00655

TABLE I. SUMMARY OF PARAMETERS FOR THE MO-PTSP CONTROLLER, WITH VALUES FOR THE 2012 PTSP CONTROLLER, HAND TUNED VALUES AND THREE RUNS OF THE CMA-ES ALGORITHM

regardless of time, fuel and damage. Finally, individuals of the same rank were ranked according to the sum of time, fuel and damage scores (with equal weight). The performance of each individual is noisy due to the random nature of MCTS, so the ranking of individuals is approximate. There are more general approaches to performing this ranking [9] but due to the noise in our problem and limited implementation time these methods were not tested. Other than our modifications to the ranking of individuals on each iteration, we used a standard implementation of the CMA-ES algorithm.

The modified CMA-ES algorithm was run for 400 iterations and produced the values presented in Table I. This was repeated three times, with each run producing different parameters (in some cases radically different). Table I gives a suggested range for each parameter, which was used to sample the initial population, but CMA-ES is able to tune the parameters outside these ranges if doing so is beneficial.

B. Performance

We measured the performance of our controller by performing 10 trials on each of the 10 maps provided with the competition starter kit. We tested six controllers in total: the PUROFVIO controller entered into the 2012 PTSP competitions, the PUROFMOVIO controller described in this paper with hand tuned parameter values, the same controller with $\beta_4 = \text{false}$ (i.e. without fuel tanks included in the route), and PUROFMOVIO with the three sets of parameters tuned by CMA-ES. Results are shown in Table II and Figure 4. Results are averaged over the 100 trials (10 trials on each of the 10 maps). For example, the "waypoints missed" figure for the 2012 controller is 0.29, meaning that this controller missed a total of 29 waypoints out of a possible 1000 (10 per run for 100 runs).

The 2012 PTSP controller achieves the fastest average time but sometimes fails to collect all waypoints and performs poorly on fuel and damage. Both hand tuned controllers are strictly dominated by one or more CMA-ES tuned controllers, demonstrating clearly the benefit of automatic parameter tuning. The parameter tuning heavily penalised any controllers which failed to complete a map so unsurprisingly the CMA-ES tuned controllers have a lower rate of waypoints missed than the hand tuned controllers.

Comparing the performance of the two hand tuned controllers, and the performance of CMA-ES run 2 to that of

Controller	Average per run: Waypoints missed	Time	Fuel	Damage
2012 PTSP controller	0.29	1394.4	632.0	444.8
Hand tuned	0.15	1951.9	256.3	461.8
Hand tuned (no fuel)	0.06	1718.1	396.4	420.2
CMA-ES run 1	0.01	1575.2	295.8	392.3
CMA-ES run 2	0.00	1547.7	355.7	412.1
CMA-ES run 3	0.01	1767.9	160.7	396.1

TABLE II. COMPARISON OF CONTROLLER PERFORMANCE

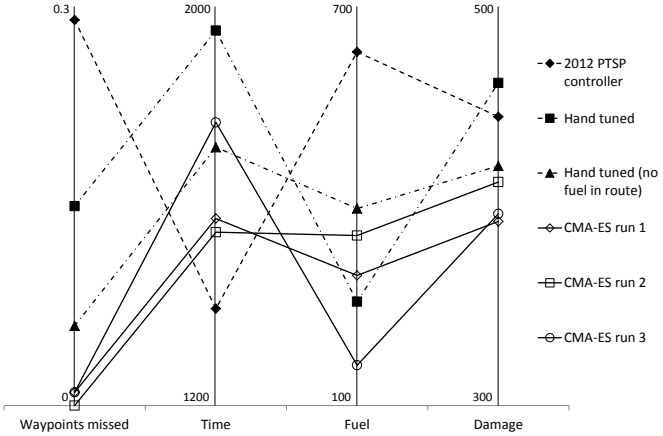

Fig. 4. Parallel coordinate plot [20] of controller performance, plotting the data in Table II. Each data point is averaged over a total of 100 trials, with 10 trials on each of the 10 starter kit maps. On each axis, lower values are better.

runs 1 and 3, it is evident that including fuel tanks in the route yields lower fuel consumption at the expense of slower times. CMA-ES run 3 achieves the lowest score for fuel consumption, at the expense of relatively slow times. The performance of CMA-ES runs 1 and 2 is similar but not identical: run 1 (which includes fuel tanks in the route) performs better on fuel and damage but run 2 (which does not includes fuel tanks) achieves slightly faster times.

All three CMA-ES tuned controllers perform well, and none dominates the others. Each of the three CMA-ES runs succeeded in finding good parameter values, although the actual values vary greatly between runs (see Table I). The three CMA-ES tuned controllers place slightly different importance on optimising for time, fuel or damage. The CMA-ES tuned parameters also performed better than the hand tuned parameters and overall the tuning method proved effective, finding robust sets of parameters which collected all waypoints more reliably than the hand tuned parameters.

Automatic parameter tuning also provides insight into the importance of some of the parameters for our controller. In earlier work [3] we exhaustively tested different values of macro-action length T and playout limit d, concluding that values of $T = 15$ and $d = 8$ performed best. The CMA-ES tuned parameters produce similar values with the macro action length $9 \leq T \leq 12$ and the playout limit $d = 7$ in each case. As was discussed in Section III-D the weight α_1 for the number of waypoints collected needs to be greater than the weight α_3 for the distance to the next route node. This is the case for all three tuned parameter sets. It is also interesting to note that the magnitude of these two parameters is not important as long as the other evaluation terms and the UCB1 exploration constant are scaled accordingly; the different set of parameters use different magnitudes, and exhibit this scaling.

There are a few anomalous parameter settings, such as negative penalty for sharp turns in run 1 and a very small penalty for damaging collisions in run 3. We would normally expect such choices to be detrimental to performance, so this may be a case of over-fitting to the training maps, where these choices perhaps do not influence the result. One weakness in our methodology is in averaging the time, fuel and damage scores across all maps. Since achieving good scores is easier on some maps than others, performance on the maps which require the most time/fuel/damage are effectively given greater weight in our evaluation. For example one of the maps is maze-like, and takes two to three times as long to complete as another map consisting mainly of open space, so the former map has two to three times more influence on the average score than the latter. This may have resulted in our tuning methodology over-fitting to the more complex maps in the already limited set of training maps.

C. Impact of panic mode

We tested versions of the hand tuned and CMA-ES tuned controllers with panic mode disabled (the 2012 controller does not have panic mode). The CMA-ES tuned controllers almost always succeed in collecting all waypoints on the map, missing at most 0.01 waypoints on average (i.e. at most 1 waypoint in 100 trials). This is true with or without panic mode: the problem when a waypoint is missed is not one of being trapped in a low quality local optimum, but rather an unlucky situation on one map in particular where the ship occasionally becomes trapped in a tight space. The ranking we use for CMA-ES heavily penalises individuals for failing to collect all waypoints, so it is unsurprising that the resulting controllers are robust in this sense.

In contrast, panic mode does improve the robustness of the controller with hand tuned parameters: this controller misses 0.15 waypoints on average per run with panic mode, but 0.84 waypoints without panic mode. The hand tuned controllers without fuel tanks in the route miss 0.06 and 0.28 waypoints respectively. This shows that panic mode can be beneficial, but careful parameter tuning can render it unnecessary.

In cases where the controller successfully collects all waypoints without panic mode, the inclusion of panic mode does not significantly change the controller's time, fuel and damage scores. Indeed we see that panic mode is rarely (if at all) invoked in these cases.

V. Conclusion

This paper presents a controller for MO-PTSP, which builds upon our world champion PTSP controller [7], [3]. Both controllers use steering based on MCTS with macro-actions, depth limiting and heuristic evaluation. The fitness function used by the MCTS steering controller is informed by a pre-planning step combining computer graphics techniques (scanline flood fill), classical TSP heuristics (multiple fragment and 3-opt) and a heuristic evaluation for route cost. This architecture was developed for the single-objective PTSP but readily applies to the more complex multiobjective problem, with appropriate modifications to the evaluations used by route planning and steering.

Automatic parameter tuning significantly improved the performance of our controller when compared to hand tuned parameters. We tried automatic parameter tuning for our entry to the 2012 PTSP competition, but did not obtain any measurable improvement over the hand tuned values. The increased number of parameters and the multi-objective nature of MO-PTSP significantly increase the difficulty of hand tuning. Our parameter tuning methodology is a new component to our controller architecture and could easily be re-used for a different game. For the MO-PTSP we ranked controllers using non-dominated sorting and other features to account for the objectives of the game. In particular controllers which failed to collect all waypoints on a map are considered to be dominated by controllers which succeeded. This resulted in our tuned controllers having a lower rate of failure compared to the hand tuned controller.

Using depth-limited MCTS to traverse the state-action graph of a single-player game can be thought of as an instance of local search. As with any local search technique, the ability of the algorithm to find a goal state can be harmed if the fitness landscape has local optima that do not correspond to goal states. We propose a system that uses the statistics collected at the root of the MCTS tree to detect when the agent is stuck in such a local optimum (i.e. in a decision cycle which does not lead to visiting the

next waypoint) and responds by restarting the search with an alternative "panic mode" fitness function. The average reward at the root is a good measure of the expected value of the principal line of play found by MCTS and can be interpreted as a measure of the quality of the lines of play found by the search. A topic for future work is to investigate other ways in which this statistic can be monitored as the search progresses, and how the behaviour of the search can be modified in response. This could be useful for domains where there is uncertainty or inaccuracy in the forward model, where good lines of play may be difficult to find using a single fitness function. We present panic mode as a first step in this direction.

For MO-PTSP specifically, there are other potential modifications to our controller. We experimented with additional macro-actions that do not apply constant thrust, for example applying thrust only on odd-numbered time steps. The rationale was to allow the controller to manoeuvre the ship more slowly but using less fuel than the full thrust macro-actions. However these additional macro-actions increase the branching factor in the tree, and we found this to be detrimental to performance. It could be that combining these actions with a search enhancement such as progressive unpruning [21] would mitigate the extra branching factor. Since the ship in MO-PTSP is subject to a drag force proportional to its velocity, travelling at faster speeds uses more fuel per unit distance. Thus we could try conserving fuel by imposing a maximum speed on the ship, with macro-actions that only apply thrust when travelling below this speed.

The hierarchical architecture of a pre-planning step combined with depth limited search during play has the potential to apply to a wide variety of games and sequential decision problems in (discretised) continuous real-time environments. Combining this architecture with automatic parameter tuning or dynamic adjustment of the fitness function to escape poor quality local optima (or both) yields AI agents that are effective at optimising fitness whilst being robust to the pathologies of a complex fitness landscape.

ACKNOWLEDGEMENTS

We thank Diego Perez and Simon Lucas for organising the MO-PTSP competition. We also thank the anonymous reviewers for their helpful comments. This work is funded by grant EP/H049061/1 of the UK Engineering and Physical Sciences Research Council (EPSRC).

REFERENCES

[1] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012.

[2] C.-S. Lee, M. Müller, and O. Teytaud, "Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 225–228, 2010.

[3] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Travelling Salesman Problem: Tree Search and Macro-Actions," *IEEE Trans. Comp. Intell. AI Games (submitted)*, 2013.

[4] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, Eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley and Sons, 1985.

[5] D. Perez, P. Rohlfshagen, and S. M. Lucas, "Monte Carlo Tree Search: Long-term versus Short-term Planning," in *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp. 219–226.

[6] D. Perez, D. Robles, and P. Rohlfshagen, "The Physical Travelling Salesman Problem Competition Website," http://ptsp-game.net/, 2012.

[7] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem," in *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp. 234–241.

[8] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.

[9] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," *Evol. Comput.*, vol. 15, no. 1, pp. 1–28, 2007.

[10] S. Remde, P. I. Cowling, K. Dahal, and N. Colledge, "Evolution of fitness functions to improve heuristic performance," in *Proc. Learn. Intell. Optim.*, Trento, Italy, 2007, pp. 206–219.

[11] E. P. K. Tsang and C. Voudouris, "Fast local search and guided local search and their application to British Telecom's workforce scheduling problem," *Oper. Res. Lett.*, vol. 20, no. 3, pp. 119–127, 1997.

[12] Atari, "Asteroids," http://www.atari.com/asteroids.

[13] H. Lieberman, "How to color in a coloring book," *ACM SIGGRAPH Comput. Graph.*, vol. 12, no. 3, pp. 111–116, 1978.

[14] J. L. Bentley, "Experiments on Traveling Salesman Heuristics," in *Proc. 1st Annu. ACM-SIAM Symp. Disc. Alg.*, 1990, pp. 91–99.

[15] S. Lin, "Computer solutions of the traveling salesman problem," *Bell Syst. Tech. J.*, vol. 44, pp. 2245–2269, 1965.

[16] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," in *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra, Eds. John Wiley and Sons, 1997, pp. 215–310.

[17] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *Euro. Conf. Mach. Learn.*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Germany: Springer, 2006, pp. 282–293.

[18] E. Balas, "The prize collecting traveling salesman problem," *Networks*, vol. 19, pp. 621–636, 1989.

[19] H. J. Berliner, "Some necessary conditions for a master chess program," in *Proc. Int. Joint Conf. Artif. Intell.*, 1973, pp. 77–85.

[20] A. Inselberg, "The plane with parallel coordinates," *Visual Comput.*, vol. 1, no. 4, pp. 69–91, 1985.

[21] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Math. Nat. Comput.*, vol. 4, no. 3, pp. 343–357, 2008.