Relazione del progetto di Computazione Naturale

Alessandro Montefusco, Vincenzo Ferrara

***Abstract*—La computazione naturale, branca dell'intelligenza artificiale, si occupa dello sviluppo di tecnologie e modelli computazionali ispirati alle metodiche che la natura ha raffinato nel corso dei millenni e sulle quali l'uomo non ha ancora trovato soluzioni adeguate. I giochi sono da sempre ottimi ambienti per testare algoritmi di computazione naturale e, più in generale, di intelligenza artificiale. Il simulatore di auto da corsa TORCS è un ideale ambiente di test in quanto può essere utilizzato anche per lo sviluppo di agenti artificiali intelligenti. In questo paper si vuole mostrare in che modo è possibile ottimizzare un set di parametri di partenza che definiscono il comportamento di un controllore virtuale di un'auto da corsa durante una simulazione in TORCS. In particolare, si parlerà dell'impiego delle due tecniche di computazione naturale utilizzate: Differential Evolution e Particle Swarm Optimization.**

I. Introduzione

Perfezionare un controllore di un'auto da corsa virtuale è un compito molto impegnativo ma allo stesso tempo affascinante: le variabili che entrano in gioco sono molteplici e riguardano sia le differenze esistenti tra i vari circuiti, sia i parametri di guida del controllore stesso. L'obiettivo di avere un controllore che sia in grado di guidare perfettamente su ogni circuito è davvero arduo da raggiungere. Tuttavia, mediante l'utilizzo di tecniche di computazione naturale è possibile ottimizzare un controllore esistente, evolvendo il suo comportamento su un set di circuiti con caratteristiche diverse (presenza di rettilinei, curve a gomito, tornanti, ecc.), in modo da ottenere risultati discreti su altri circuiti. L'ambiente di esecuzione utilizzato per le simulazioni è TORCS (*The Open Racing Car Simulator*), un simulatore di gare automobilistiche 3D con diversi tracciati. In questo documento verrà fatta prima una analisi sull'ambiente di simulazione e le diverse variabili in gioco, dopodiché si passerà ad analizzare le tecniche di computazione naturale utilizzate, la funzione di fitness impiegata per l'ottimizzazione, ed infine verranno illustrati gli esperimenti effettuati e i risultati ottenuti.

II. TORCS

Per lo sviluppo e la realizzazione del controllore è stato utilizzato il simulatore TORCS, una piattaforma utilizzata largamente per competizioni di AI e per la ricerca. La piattaforma permette di gareggiare in diverse modalità: Quickrace, Practice, Championship race, Endurance race, DTM race e Non championship race. Per il progetto in questione la modalità scelta è la corsa individuale, senza la presenza di avversari in pista.
Prima di partire con l'ottimizzazione è stata fatta una attenta analisi sull'ambiente stesso. Una prima considerazione è stata fatta sui tempi di esecuzione delle simulazioni: ogni gara di 2 giri dura circa 4-5 minuti (a seconda della difficoltà e lunghezza del tracciato), e risulta quindi molto oneroso utilizzare il simulatore per la ricerca. Per tale ragione, TORCS introduce la possibilità di avviare l'esecuzione in *Text-mode*, in cui non è presente la grafica. Tale configurazione permette di abbattere notevolmente i tempi di simulazione (ora nell'ordine dei secondi), ed è fortemente consigliata per l'esecuzione di esperimenti in batch mode.
Un ulteriore attenzione è stata posta sull'analisi dei sensori e degli attuatori che regolano la guida del controllore (Tabella I e Tabella II).
Infine, si è notato che bisogna tenere in considerazione che il Server attua dei controlli sui limiti di tempo della gara: nel caso in cui un'automobile non termini in un tempo prestabilito, la macchina viene prelevata dalla pista e la connessione viene interrotta. Inoltre, i tempi di corsa possono essere fortemente influenzati anche dal danno che l'auto accumula durante la gara: maggiore è il danno, minore sarà la velocità del veicolo. Nel corrente caso per scelte di progetto i danni non stati considerati, settandoli a zero mediante una apposita opzione messa a disposizione da TORCS, e concentrandosi sulla guida.

A. Sensori

Il controllore percepisce l'ambiente attraverso una serie di sensori che forniscono informazioni sia sull'ambiente di gioco circostante (pista, avversari, velocità, angoli di direzione dell'auto, ecc.), sia sullo stato attuale della gara (il tempo sul giro corrente, la posizione in gara, giri del motore, danni riportati, ecc.). Le letture di questi sensori avvengono ogni 20 millisecondi e vanno a determinare la guida del controllore.

SENSORE	DESCRIZIONE
angle	Angolo tra la direzione dell'auto e quella dell'asse del tracciato.
curLapTime	Tempo trascorso nel giro corrente.
damage	Danno corrente dell'auto.
distFromStart	Distanza dal via dell'auto.
distRaced	Distanza percorsa dall'auto dall'inizio della gara.
focus	Vettore di 5 sensori che restituiscono la distanza tra l'auto e i limiti del tracciato.
fuel	Valore corrente del carburante.
gear	Marcia attualmente in uso.
lastLapTime	Tempo sull'ultimo giro.
opponents	Vettore di 36 sensori, ognuno copre 10 gradi e restituisce la distanza dall'avversario più vicino.
racePos	Posizione in gara rispetto alle altre auto.
rpm	Numero di giri al minuto del motore dell'auto.
speedX	Velocità dell'auto lungo l'asse delle X.
speedY	Velocità dell'auto lungo l'asse delle Y.
speedZ	Velocità dell'auto lungo l'asse delle Z.
track	Vettore di 19 sensori che restituiscono la distanza tra l'auto e i limiti del tracciato.
trackPos	Distanza tra la direzione dell'auto e l'asse del tracciato.
wheelSpinVel	Vettore di 4 sensori che restituiscono la velocità di rotazione di ciascuna ruota.
z	Distanza del centro di massa dell'auto dalla superficie del tracciato lungo l'asse Z.

TABLE I: Sensori dell'auto.

B. Attuatori

L'auto viene controllata mediante una serie di attuatori quali: il volante, l'acceleratore, il freno e il cambio. Inoltre, è disponibile una meta-azione per richiedere un riavvio della gara al Server.

Attuatori	Descrizione
accel	Acceleratore
brake	Freno
clutch	Frizione
gear	Marcia
steering	Valore di sterzata
focus	Direzione dell'auto
meta	Parametro per controllare il riavvio della gara

TABLE II: Attuatori dell'auto.

III. Starting point - CarSim

Il controllore sub-ottimo di partenza è CarSim. I parametri di default sono forniti dallo stesso sviluppatore nel file *default_parameters*, il quale fornisce anche il codice del *driver* del controllore; il tutto è messo a disposizione sul sito ufficiale di TORCS [1]. Alcuni di questi parametri non sono utilizzati nella guida (*meta parameters*), altri sono fissati all'interno del codice, quindi non soggetti a variazioni e pertanto non saranno oggetto del nostro studio (Tabella III).

Parametri
syint
syslp
clutch_release
turnsetup
S2ang
senslim
accinc
sensang

TABLE III: Parametri non influenti sulla guida.

Definiti i parametri da evolvere (Tabella IV e Tabella V), si è passati all'analisi dei range entro i quali tali parametri dovevano variare. Siccome il controllore è "sub-ottimo", ogni parametro è stato fatto variare in un range $\pm 50\%$ rispetto al valore di default, in modo da ricercare l'ottimo in un intorno di quel controllore. Inoltre, per poter condurre degli esperimenti confrontabili, è importante che la macchina parta sempre dalla stessa posizione nel tracciato per evitare di condizionare il corso della gara.
Come accennato nel paragrafo precedente, le simulazioni vengono condotte in *Text-mode* e per poter ottimizzare ancora di più i tempi, sono stati stanziati, all'inizio di ogni run, due Server per eseguire due gare in parallelo su due porte diverse e, quindi, effettuare delle valutazioni più veloci e in numero più elevato. Ad ogni Server è associato un Client (il driver). Ogni coppia Client-Server rappresenta una gara su uno dei due tracciati scelti per l'evoluzione del controllore.
Grazie a queste considerazioni iniziali è stato possibile valutare 20 individui su due piste diverse in parallelo, per un totale di 40 simulazioni in media ogni 5 minuti.

IV. Tecniche

In questo paragrafo verranno introdotte le tecniche utilizzate per la risoluzione del problema descritto: un problema di ottimizzazione basato su valori reali. Tra le possibili tecniche di computazione naturale, presentate durante il corso, ci sono: *ImmunoComputing* e *Ant Colony*, la prima scartata in quanto è usata principalmente per compiti di pattern recognition, mentre la seconda è utilizzata per risolvere problemi di ottimizzazione ma con valori discreti: onde evitare modifiche al codice per adattarlo al nostro problema, si è preferito scartare questa tecnica. *NeuroComputing* con un apprendimento basato su rinforzo è una tecnica molto interessante e sarebbe stata una buona scelta, tuttavia non è stata presa in considerazione per la limitatezza delle risorse:

Parametri	Limite inf	Limite sup
dnsh3rpm	4912.780198	9123.734654
upsh6rpm	8340.15191	15488.853547
dnsh2rpm	4419.127724	8206.951488
upsh4rpm	6668.706859	12384.74131
dnsh1rpm	2809.45253	5217.554698
dnsh4rpm	5201.217344	9659.403638
dnsh5rpm	5238.429763	9728.512416
upsh5rpm	6756.951019	12548.623321
upsh2rpm	6669.926259	12387.005909
upsh3rpm	6663.42871	12374.939034

TABLE IV: Parametri per il controllo delle marce.

Parametri	Limite inf	Limite sup
consideredstr8	0.007023	0.013043
str8thresh	0.100686	0.186989
str8thresh	0.100686	0.186989
safeatanyspeed	0.000896	0.001664
wwlim	3.140934	5.833163
s2sen	2.103102	3.905762
seriousABS	19.539464	36.287576
obviousbase	66.715006	123.899297
offroad	0.700186	1.300345
stst	346.085517	642.730246
fullstmaxsx	14.049573	26.092065
ignoreinfleA	7.555491	14.031626
stC	230.407561	427.899756
spincutslp	0.035998	0.066854
slipdec	0.012633	0.023462
pointingahead	1.537512	2.855379
oksyp	0.046107	0.085627
spincutclip	0.07178	0.133305
sxappropriatest1	11.258289	20.908251
clutchslip	63.238705	117.443309
obvious	0.939726	1.745206
backontracksx	49.595437	92.105811
clutchspin	35.203725	65.378346
fullstis	0.573912	1.065836
brake	7.7e-05	0.000143
carmin	23.708146	44.029414
sycon2	0.700167	1.300311
s2cen	0.357942	0.66475
sycon1	0.450047	0.835802
carmaxvisib	1.604301	2.979416
sxappropriatest2	0.386414	0.717626
skidsev1	0.403632	0.749602
wheeldia	0.598799	1.112056
brakingpacefast	0.726839	1.349844
spincutint	1.221491	2.268483
st	482.968806	896.942068
brakingpaceslow	1.569796	2.915335
sortofontrack	1.052848	1.955289
steer2edge	0.665148	1.235275
backward	1.168457	2.169992

TABLE V: Parametri per il controllo della pista e dell'auto.

fare un fine tuning di una rete neuronica richiede molto tempo soprattutto se non si posseggono le adeguate risorse computazionali. Non è stato scelto il *Genetic Programming* in quanto richiede una attenta analisi per definire quale siano gli elementi principali da inserire nei due set (terminal e non-terminal); questa analisi risulterebbe molto complessa e avrebbe portato via molto tempo rispetto a quello a disposizione. Ciò non toglie che tale tecnica non la si possa utilizzare in futuro per migliorare le prestazioni del controllore. Le alternative che sono sembrate più adatte, in termini di tempo d'esecuzione e di problema da risolvere, sono: *Differential Evolution* e *Particle Swarm Optimization*.

A. Differential Evolution

DE è un algoritmo evolutivo di tipo stocastico, pensato per risolvere problemi di ottimizzazione che hanno valori reali, utilizzando un processo di ricerca per migliorare iterativamente una popolazione di soluzioni. L'evoluzione è di tipo generazionale, cioè si ha un forte cambiamento della popolazione tra una generazione e un'altra. La potenza di questo algoritmo risiede nel fatto che ci sono soltanto 3 iperparametri da dover settare: *NP*, grandezza della popolazione; *F*, operatore genetico di mutazione; *CR*, operatore genetico di ricombinazione. Il parametro CR controlla l'influenza che ha un genitore nella generazione della prole: regola con quale probabilità si va a selezionare una componente dal vettore donor (o anche mutant vector) o dall'individuo di di partenza. Il parametro F permette di spostarci in un qualsiasi punto dello spazio di ricerca, in modo da non rimanere bloccati in un minimo locale. La mutazione, però, deve essere ben pesata: se è vero che mutando non abbiamo stagnazione, è anche vero che mutazioni troppo grandi inducono una ricerca quasi casuale. Per settare questi due iperparametri si è scelto di utilizzare la variante *Self-Adaptive DE* che è in grado evolverli insieme agli individui della popolazione. L'unico iperparametro settato è la dimensione della popolazione. In generale, DE risolve problemi complessi anche con un numero molto basso di individui. Storn e Price suggeriscono di usare una popolazione nel range $[5D; 10D]$, dove D è la dimensionalità del problema. A causa delle scarse risorse temporali, si è scelto di utilizzare una popolazione di 20 individui.
Inizializzata la popolazione, per ogni individuo, di generazione in generazione, si applicano gli operatori genetici e l'operatore di selezione.
L'operatore genetico di mutazione crea il cosidetto donor vecotr; SADE implementa diverse varianti per la generazione di questo vettore. Tra queste sono state selezionate le tre varianti che, secondo la letteratura, presentano una più rapida convergenza:

- *DE/rand/1/bin* dimostra una velocità di convergenza più lenta a fronte di una migliore ricerca nello spazio delle soluzioni. Tale variante costruisce il donor vector selezionando tre individui random dalla popolazione:
$$v_i(t+1) = x_{r1}(t) + F\big(x_{r2}(t) - x_{r3}(t)\big) \tag{1}$$
- *DE/best/2/bin* dimostra una velocità di convergenza maggiore, tuttavia si ha più probabilità di rimanere bloccati in un ottimo locale se si ha a che fare con problemi multimodali. Tale variante costruisce il donor vector basandosi sull'individuo migliore della popolazione corrente e usando due diversi *difference vector*:
$$v_i(t+1) = x_b(t) + F\big(x_{r1}(t) - x_{r2}(t)\big) + F\big(x_{r3}(t) - x_{r4}(t)\big) \tag{2}$$
- *DE/rand-to-best/1/bin* dimostra anch'esso una velocità di convergenza maggiore ma costruisce il donor vector basandosi su una combinazione delle due varianti precedenti:
$$v_i(t+1) = x_i(t) + F\big(x_b(t) - x_i(t)\big) + F\big(x_{r1}(t) - x_{r2}(t)\big) \tag{3}$$

A questo punto verrà applicato l'operatore genetico di ricombinazione per la generazione di un nuovo individuo. Ci sono due schemi di crossover differenti: *Bin* ed *Exp*. Con Bin si intende che l'i-esimo parametro assegnato al *target vector* viene preso o dall'individuo corrente o dal donor vector con una certa probabilità. Con Exp si scelgono due punti random all'interno del donor vector; gli N parametri compresi tra i due punti vengono trascritti nel target vector, mentre i restanti parametri vengono selezionati dall'individuo corrente. Dal momento che le risorse temporali erano limitate, si è optato direttamente per lo schema Bin, senza adoperare un confronto, dal momento che in letteratura viene riportato come schema che mediamente assicura migliori proprietà di convergenza.
Infine, l'operatore di selezione, basandosi sui valori della fitness, selezionerà, tra l'individuo corrente e il target vector, l'individuo migliore: ciò significa che, generazione dopo generazione, la popolazione può sia migliorare che rimanere costante, ma mai deteriorare.

B. Particle Swarm Optimization

PSO è un metodo computazionale di tipo stocastico che ottimizza un problema iterativamente, cercando di migliorare una soluzione in relazione a una determinata misura di qualità. PSO risolve un problema avendo una popolazione di soluzioni candidate, denominate *particelle*, e spostando queste particelle nello spazio di ricerca secondo semplici regole basate sulla posizione e la velocità della particella. La ricerca consiste nel trovare la migliore posizione possibile (fitness migliore). Il movimento di ogni particella è influenzato da due informazioni: una locale, proveniente dalla sua esperienza, e una globale, proveniente dall'esperienza di tutte le altre particelle. L'informazione globale dello sciame garantisce che non ci si blocchi intorno ad un minimo locale. Ad ogni iterazione ogni particella viene spostata mediante un vettore velocità che dipende sia dalla miglior posizione trovata dalla particella stessa (p^{best}), sia dalla miglior posizione trovata dai suoi vicini (g^{best}). Grazie all'interazione di questi semplici agenti, riesce ad emergere un comportamento dello sciame molto più complesso che porta alla risoluzione del problema. PSO non ha processi di mutazione o crossover espliciti. Tuttavia, i processi di aggiornamento di velocità e posizione possono essere considerati come un crossover implicito grazie alle informazioni di p^{best} e g^{best}. Infine, sebbene non vi sia alcuna selezione esplicita, l'attrazione delle particelle verso

il g^{best} funge da meccanismo di selezione implicito in quanto influenza la velocità di tutte le altre particelle.
Per poter comunicare tra loro, le particelle devono essere connesse mediante un grafo di interazione. La topologia dello sciame definisce il sottoinsieme di particelle con cui ciascuna particella può scambiare informazioni. La topologia che usa l'algoritmo canonico è quella *globale*, che consente a tutte le particelle di comunicare con tutte le altre, quindi l'intero sciame condivide la stessa g^{best}. Tuttavia, questo approccio potrebbe portare lo sciame ad essere bloccato in un ottimo locale, pertanto è stata analizzata in parallelo anche la topologia *locale*, in cui le particelle condividono informazioni solo con un sottoinsieme di particelle, nel caso in questione, ogni particella è collegata ad altre 4 particelle.
Un problema da tenere in considerazione su PSO è la velocità delle particelle: essendo strettamente dipendente dall'informazione locale e globale, più si è vicini all'ottimo più la particella deve andare piano. Le particelle lontane da p^{best} e g^{best} tendono a produrre aggiornamenti molto grandi del vettore velocità: questo comportamento può portare all'instabilità del sistema, rendendo difficile allo sciame cercare regioni di alta qualità. Per limitare queste forte oscillazioni e, quindi, controllare il vettore velocità, si è utilizzata la variante con *fattore di costrizione*:

$$\overrightarrow{v_i}(t+1) = K\big[\overrightarrow{v_i}(t) + U(0,\phi_1) \otimes \big(\overrightarrow{p_i}^{best}(t) - \overrightarrow{x_i}(t)\big) + \\ + U(0,\phi_2) \otimes \big(\overrightarrow{g_i}^{best}(t) - \overrightarrow{x_i}(t)\big)\big] \tag{4}$$

Dove *K* è il fattore di costrizione:

$$\frac{2}{\left|2 - c - \sqrt{c^2 - 4c}\right|} \quad c = \phi_1 + \phi_2 \tag{5}$$

A causa delle scarse risorse temporali, non è stato possibile fare un fine tuning degli iperparametri, i cui valori sono stati settati in base ai valori suggeriti in letteratura: $\phi_1 = \phi_2 = 1.49618$.

V. Funzione di fitness

In generale, la funzione di fitness è una funzione in grado di associare ad ogni soluzione un certo valore che sta ad indicare la qualità di quella soluzione in base al problema che si vuole risolvere. Definire la giusta funzione obiettivo per calcolare la fitness è un elemento fondamentale che sta alla base di tutte le tecniche di computazione naturale. Per definire, quindi, una funzione ottimale sono state condotte diverse simulazioni con il controllore sub-ottimo di CarSim. I problemi evidenziati nel corso delle analisi sono molteplici e di diversa natura.
I problemi più evidenti sono stati individuati nella *velocità di crociera*, velocità che si raggiunge al regime di coppia massimo con l'ultimo rapporto disponibile [1]; nel *cambio della marcia*; nella *tenuta di strada*; nelle *brusche frenate* prima di entrare in curva e i lunghi *fuori pista* dopo una curva non riuscita. Alla luce dei problemi riscontrati si è deciso di implementare due funzioni obiettivo diverse: una prima che tenesse conto del solo andamento della velocità e una seconda che tenesse conto sia dell'andamento della velocità sia della posizione dell'auto rispetto all'asse del tracciato.

[1]Possiamo definire quindi la velocità di crociera come la massima velocità costante che permette a tutti gli organi dell'auto di lavorare in modo corretto senza essere sfruttati permettendo di risparmiare tempo e allo stesso modo carburante e manutenzione.

A. Funzione di fitness φ_1 - Velocità

Nell'analizzare la prima funzione di fitness φ_1 sono stati presi in considerazione 3 sensori messi a disposizione dell'auto: *speedX*, *speedY* e *speedZ*. Tali sensori restituiscono le velocità lungo i 3 assi X, Y e Z ogni 20ms. A tal proposito è giusto sottolineare che la fitness calcolata tiene conto dei valori ottenuti dai sensori e non è ottenuta dalla relazione tra lo spazio percorso e il tempo impiegato a percorrerlo. Dal punto di vista pratico, quindi, ad ogni giro vengono immagazzinate tutte le letture della velocità lungo gli assi, calcolato il modulo della velocità e, infine, valutata la velocità media durante tutto il tracciato. Il problema è stato definito come un problema di minimizzazione, di conseguenza si considera l'inverso di tale velocità:

$$\varphi_1(x_i) = \frac{N}{\sum_{i=0}^{N} \sqrt{{v_x}_i^2 + {v_y}_i^2 + {v_z}_i^2}} \tag{6}$$

Dove x è l'*individuo considerato*, N il *numero di step* necessari per completare un giro.

B. Funzione di fitness φ_2 - Posizione e velocità

Come precedentemente introdotto, dall'analisi effettuata su CarSim è emerso un controllore che tendeva a stare troppo fuori dalla pista e a non seguire l'asse del tracciato. Per evitare di far emergere un comportamento che tenesse conto soltanto della velocità, producendo un controllore che tagliasse le curve, si è giunti alla definizione di una seconda funzione obiettivo φ_2 che tenesse conto non solo della velocità media lungo il tracciato, bensì anche della guida e della posizione sulla pista. In questo modo l'obiettivo fissato diventa non solo la conclusione del giro in tempi veloci ma anche una guida senza fuori pista o tagli delle curve. Si vuole far emergere, quindi, un comportamento quanto più simile ad un'auto da corsa reale, in quanto tutti i fuori pista sono considerati delle penalità.
Per misurare la posizione dell'auto nella pista, è stato utilizzato il sensore *trackPos*, che permette di conoscere la distanza tra la direzione dell'auto e l'asse del tracciato. I range di corretto comportamento sono $[-1, 1]$, cioè quando l'auto è sul bordo sinistro (o destro) della carreggiata e 0 quando è sull'asse. Tuttavia, dopo una attenta analisi sui valori di questo sensore, è stata fatta una distinzione tra due possibili casi:

1) l'auto percorre dei tratti con una parte della carrozzeria (con sole 2 ruote) fuori dal tracciato. Tale situazione viene monitorata aggiornando un opportuno contatore, detto t_b[2]. Non rappresentando un tempo bensì quante volta l'auto è stata sul bordo pista, è necessario processarlo per portarlo nel dominio del tempo (secondi).

[2]range dei valori di trackPos $[0.95, 1.2]$.

2) L'auto è completamente fuori pista. Anche in questo caso si fa uso di un contatore, detto t_o[3], che dovrà necessariamente essere processato per trasformarlo in un'unità di misura temporale.

La funzione obiettivo risultante è la seguente:

$$\varphi_2(x_i) = \varphi_1(x_i) + \left(K_b \cdot t_b + K_o \cdot t_o\right) \tag{7}$$

Le due penalità introdotte, K_b (penalità per il fuori bordo) e K_o (penalità per il fuori pista) sono due costanti opportunamente pesate, in quanto percorrere una porzione di tracciato fuori pista è molto più penalizzante rispetto al percorrere il tracciato sui bordi. Tali costanti sono state raffinate in seguito ad una analisi dei risultati conseguiti sui vari esperimenti.

- Valori più piccoli di K_b e K_o facevano emergere un comportamento simile a quello emerso usando la funzione φ_1: controllori che tagliavano le curve del tracciato per andare più veloci anche laddove non era possibile (attraversando lo sterrato).
- Valori più grandi di K_b e K_o facevano evolvere dei controllori con una guida troppo dolce e che non traevano vantaggio in curve laddove era possibile tirare dritto, rimanendo in pista, senza alcuna sterzata (Figura 1).

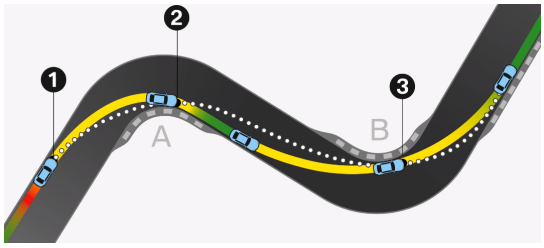

Fig. 1: Comportamento desiderato.

Tuttavia, per non penalizzare troppo gli individui iniziali, tali penalità vengono moltiplicate per una quantità definita *Adaptation factor*. L'Adaptation Factor (AF) considera la generazione corrente ($g(t)$)) e le generazioni totali (N_{gen}). In questo modo si definisce una tolleranza sugli individui a seconda della generazione in esame. Quindi, le prime generazioni vengono penalizzate meno affinché gli individui possano evolvere ma, col progredire delle generazioni, gli individui che non mostrano un corretto comportamento verranno sempre scartati.

$$AF = 1 + \frac{g(t)}{N_{gen}} \tag{8}$$

$$K_b = 0.05 \cdot AF \qquad K_o = 0.5 \cdot AF \tag{9}$$

C. Fitness su più circuiti

Per validare le due tecniche utilizzate, ogni controllore è stato ottimizzato su due tracciati con caratteristiche non troppo differenti tra loro: evolvere un controllore su circuiti nettamente diversi (tornanti in uno e curve larghe in un altro, assenza o presenza di rettilinei, ecc) porterebbe ad un controllore che non migliorerebbe mai, poiché migliorare su una pista significherebbe peggiorare sull'altra e viceversa. L'obiettivo prefissato è quindi quello di ottenere un controllore che guidi mediamente bene su più circuiti, senza specializzarsi troppo su uno. Per tale motivo, ogni simulazione consiste nel far correre ogni individuo su due piste in parallelo; la fitness dell'individuo è data dalla media delle due fitness calcolate sui due tracciati. La fitness di ogni tracciato è la somma delle fitness sui singoli giri. I giri effettuati sono due per ogni tracciato: nel primo giro le prestazioni dipendono anche dal fatto che il controllore parte da fermo; invece, tutti i giri dopo il secondo presentano prestazioni identiche. Così facendo, si garantisce che il controllore abbia buone prestazioni sul primo giro e che abbia le migliori dal secondo giro in poi.
Tra i diversi tracciati proposti dalla piattaforma TORCS sono state scelte due piste per l'addestramento *CG-Speedway n.1* (Fig. 2) e quattro diverse piste per testare che il controllore non si sia specializzato troppo, come nei successivamente, nella sezione Risultati, illustrato.

Fig. 2: A sinistra CG Speedway n.1 e a destra Forza.

Malgrado le sperimentazioni condotte, non è sufficiente ad avere prestazioni ottimali su tutti i circuiti, poiché per ottenere tali risultati bisognerebbe addestrare il controllore su più circuiti e cercare di generalizzare quanto più possibile: ciò richiede tempi computazionali molto elevati.

VI. Esperimenti

Tutti gli esperimenti sono stati condotti utilizzando le tecniche e le funzioni di fitness sopra citate. Per ogni sperimentazione sono state eseguite 5 trials, variando l'inizializzazione degli algoritmi con 5 diversi semi, ottenuti attraverso un generatore di numeri pseudocasuali. L'evoluzione del controllore è stata divisa in due step: in una prima fase si è cercato di capire quale tra le varianti proposte, per ognuna delle due tecniche, avesse una più rapida convergenza verso un valore di ottimo. In questa prima evoluzione è stata considerata una popolazione di 20 individui e un numero di generazioni pari a 50. Dopodiché, sulla base di una analisi statistica, è stata fatta una seconda evoluzione prendendo in considerazione, per ognuna delle due tecniche, solo la variante giudicata migliore rispetto le altre. Tale considerazioni sono state fatte anche in merito alla limitatezza delle risorse temporali. In questa seconda evoluzione si è considerata una popolazione di 20 individui, ma un numero di generazioni pari a 30, in quanto si è notato, alla fine della prima evoluzione, che i valori della fitness nelle ultime generazioni in alcuni casi decrescevano molto lentamente, in altri erano stabili.

A. La definizione del problema

Entrambi gli algoritmi sono basati su un User Definied Problem (*UDP*). La definizione del problema consiste nella

[3]valori di trackPos maggiori di 1.2.

implementazione delle due funzioni cardine: *fitness()* e *get_bounds()*. La prima funzione è il cuore dell'algoritmo su cui si basa l'operatore di selezione attraverso l'invocazione delle fitness precedentemente definite e che permette l'evolvere della popolazione. La seconda funzione permette di definire i range di definizione di ogni parametro da cui un individuo è composto.
Tutte le funzioni e le implementazioni delle tecniche di computazione utilizzate sono state implementate in Python mediante l'utilizzo della libreria *Pygmo* [3].

B. Evoluzione I

Nella prima evoluzione si è partiti dal controllore sub-ottimo CarSim, ricercando soluzioni migliori in quell'intorno, ed impostando il range di variazione dei parametri di $\pm 50\%$ del loro valore iniziale. I parametri sottoposti all'evoluzione sono stati definiti precedentemente: ogni individuo della popolazione è un vettore di 49 parametri di guida, dunque la dimensionalità del problema è pari a 49.

C. Evoluzione II

Dalla prima evoluzione si è notato che diversi parametri si bloccavano sui bounds definiti in precedenza. Per tale ragione si è pensato di variarli ulteriormente del $\pm 20\%$ rispetto ai valori su cui si erano assestati, per controllare se le assunzioni iniziali erano troppo restringenti. Come descritto in precedenza, le analisi statistiche hanno evidenziato che alcune varianti davano performance migliori e convergevano più rapidamente verso una soluzione ottima, per cui si è scelto di considerare solo tali varianti. La popolazione di partenza di questa seconda evoluzione è caratterizzata in parte dagli individui migliori ottenuti dalla evoluzione precedente, e in parte da individui random. In questo modo si definisce l'intorno entro cui cercare le nuove soluzioni.

D. Analisi statistica

I dati ottenuti, sono stati analizzati mediante la piattaforma web **STAC** [2], che permette di fare diversi test statistici, sia parametrici che non parametrici. È stato utilizzato il test non parametrico *Friedman-Aligned-Ranks* con un livello di significatività α di 0.05. Tale test permette di fare una analisi post-hoc per determinare se l'ipotesi nulla H_0[4] possa essere rigettata oppure no. L'analisi post-hoc è stata condotta tramite due test: *Bonferroni-Dunn* e *Holm*.

VII. Risultati

A. Confronto rand/1/bin, best/2/bin e rand-to-best/1/bin per SADE

Al termine della prima evoluzione si è condotta una analisi statistica sui risultati ottenuti per le tre varianti utilizzate con SADE. Osservando l'andamento medio dei valori di φ_1 sulle 50 generazioni si evince che la variante best/2/bin ha una più rapida convergenza, come mostra la Figura 3. Tale risultato risulta essere prevedibile, in quanto la strategia basata su due vettori differenza, invece che uno, causa una mutazione più significativa che va ad ampliare lo spazio di ricerca, evitando di rimanere bloccati in qualche ottimo locale. Inoltre, la mutazione usata da questa variante, oltre che individui puramente casuali, tiene conto anche del miglior individuo all'interno della popolazione (come mostrato dall'equazione 2).

[4]Ipotesi nulla (H0): la media dei risultati delle varianti utilizzate è la stessa.

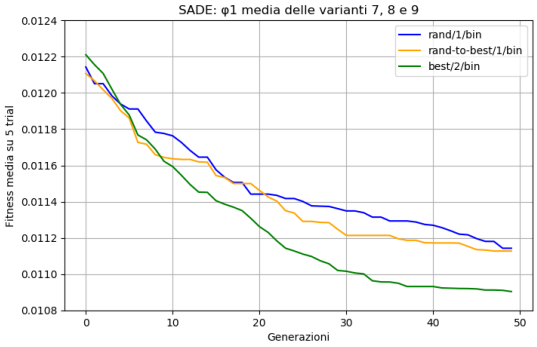

Fig. 3: SADE: fitness media delle 3 varianti a confronto.

Per quanto riguarda i risultati di φ_2, come si evince dalla Figura 4, tutte e tre le varianti hanno una buona e rapida convergenza, con leggere differenze tra i risultati. Tuttavia, lo studio dell'analisi statistica mostra che la variante migliore è *rand-to-best/1/bin*: nelle prime generazioni si può osservare come tale variante risulta essere nel mezzo delle altre due ma, nelle ultime generazioni, raggiunge valori migliori rispetto le altre. Inoltre, rispetto a φ_1, l'andamento medio di φ_2 sembra assestarsi dalla generazione 40 in poi. Per essere certi di non essere rimasti bloccati in un ottimo locale, sono state condotte altre sperimentazioni nel corso della seconda evoluzione.

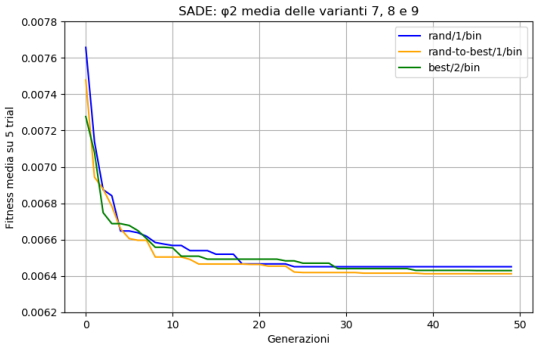

Fig. 4: SADE: fitness media delle 3 varianti a confronto.

B. Confronto global best e local best per PSO

Anche per PSO, al termine della prima evoluzione, si è fatto uno studio statistico sull'andamento medio della fitness, per le due varianti, nel corso delle 50 generazioni. Osservando l'andamento di φ_1, come mostra la Figura 5, la variante local best ha una più rapida convergenza verso l'ottimo e si dimostra essere la migliore.

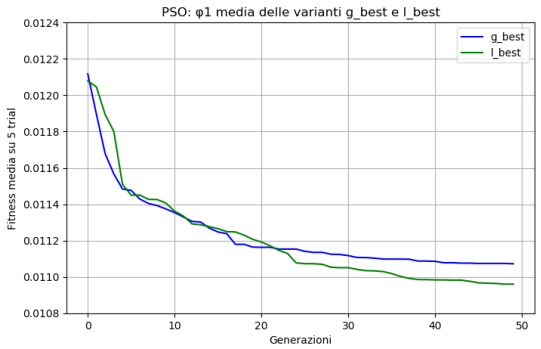

Fig. 5: PSO: fitness media delle 2 varianti a confronto.

Per quanto riguarda i risultati di φ_2, come si evince dalla Figura 6, la variante global best ha una più rapida convergenza verso l'ottimo, raggiungendo il valore minimo poco dopo la decima generazione, per poi arrestarsi del tutto. Al contrario, l'andamento della local best, seppure all'inizio sembra avere una convergenza più lenta, dimostra di raggiungere un valore della fitness mediamente più basso e di continuare l'evoluzione ben oltre la cinquantesima generazione, ragion per cui, anche considerando i risultati statistici, dimostra di essere la variante migliore.

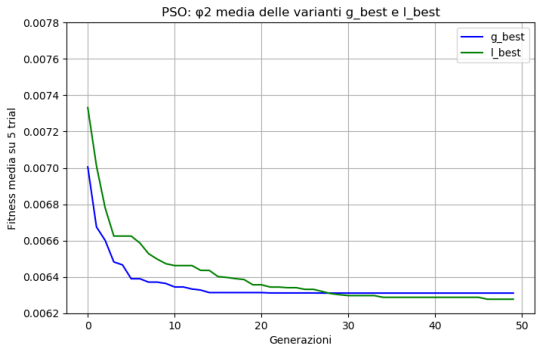

Fig. 6: PSO: fitness media delle 2 varianti a confronto.

C. Evoluzione 2

Nel corso dell'evoluzione 2, con un numero di generazioni pari a 30, si riscontrano dei miglioramenti nei controllori, sia per PSO che per SADE. Per SADE, mentre i valori della fitness associati alla funzione obiettivo φ_1 continuano a decrescere, anche se molto lentamente (Figura 7); i valori della funzione φ_2 sono costanti, nonostante un cambiamento dei bounds (Figura 8), segno che non c'è stata alcuna evoluzione. Mentre invece, per PSO, l'andamento della fitness continua a decrescere per entrambe le funzioni obiettivo utilizzate, anche se molto lentamente.

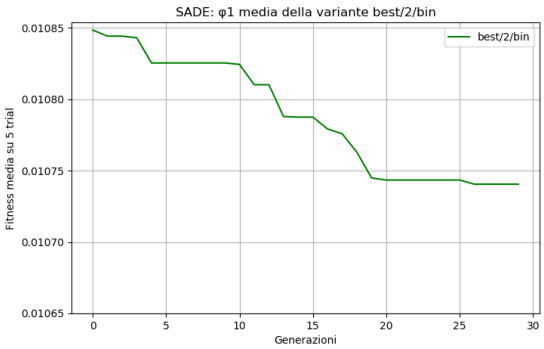

Fig. 7: SADE: evoluzione con la variante 9 per la fitness φ_1.

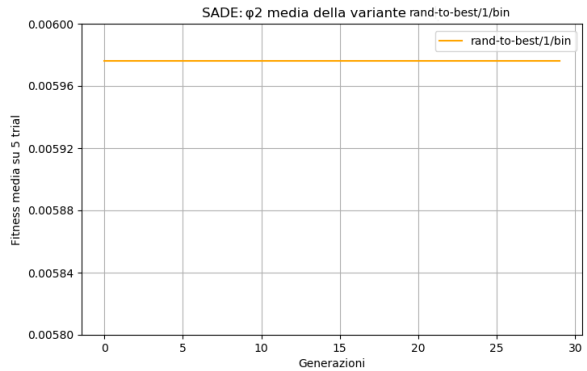

Fig. 8: SADE: evoluzione con la variante 8 per la fitness φ_2.

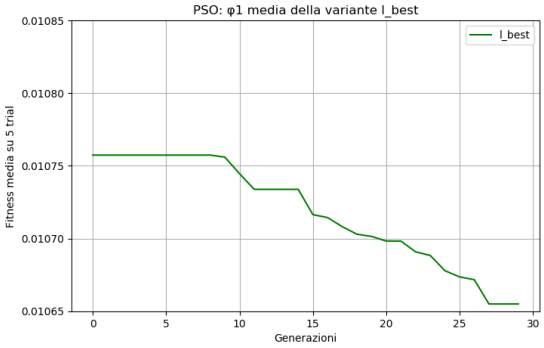

Fig. 9: PSO: andamento medio di φ_1 per la variante local_best.

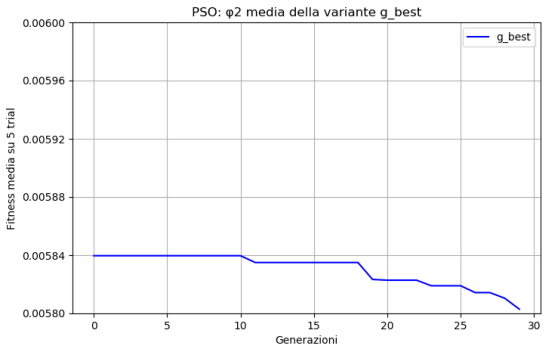

Fig. 10: PSO: andamento medio di φ_2 per la variante global_best.

D. *Fitness φ_1 vs Fitness φ_2*

Per entrambe le tecniche, SADE e PSO, l'andamento medio di φ_1 presenta una convergenza molto più lenta ma che decresce lungo tutto il corso delle due evoluzioni. Risulterebbe, dunque, ragionevole una ulteriore indagine per verificare di quanto effettivamente il controllore sia in grado di migliorare. Al contrario, l'andamento medio di φ_2 ha una più rapida convergenza, portando subito a soluzioni nettamente migliori, e che si assesta, o comunque varia di molto poco, dopo la generazione 35. Il motivo dei due diversi andamenti lo si riscontra immediatamente dai comportamenti che esibiscono i controllori.
Con la funzione φ_1, il controllore mostra una guida molto più aggressiva rispetto a CarSim (in termini di velocità), non tenendo però conto dei fuori pista. Tale comportamento, in un ambiente reale in cui non è lecito tagliare le curve, sarebbe fortemente penalizzato. Quindi, con l'aumentare delle generazioni, il controllore impara a guidare sempre più velocemente, con una tenuta di strada nettamente superiore a CarSim, ma non guidando perfettamente in curva. Un tale controllore risulta molto efficiente per quelle piste in cui sono prevalenti i rettilinei e del tutto assenti curve come le "chicane".
Con la funzione φ_2 il controllore mostra una guida nettamente più veloce rispetto a CarSim, con una buona tenuta di strada, ma con dei picchi di velocità che sono inferiori o nell'intorno di quelli raggiunti dal controllore evoluto con la prima fitness. Il comportamento emergente è quello di un controllore in grado di: riconoscere la presenza di una curva e rallentare prontamente; spingere al massimo sui rettilinei; tagliare le curve in cui è possibile, però, mantenere la pista. Tuttavia, a causa dell'assenza di curve troppo strette durante l'evoluzione, nessuno dei controllori è in grado di ottenere ottimi risultati su una pista che presenta molti tornanti ma, grazie alla buona tenuta di strada, i tempi sui giri sono migliori rispetto al controllore CarSim.

E. *SADE vs PSO*

Per confrontare nel modo migliore le performance delle due tecniche è stato utilizzato, anche in questo caso, un test statistico. Il numero di trial considerati è pari a 5: per ottenere una analisi più robusta e significativa sarebbe stato opportuno considerare un numero maggiore di trial, cosa che non è stata possibile per i limiti temporali. I test di ipotesi effettuati sono due: confronto tra SADE e PSO sia per i risultati ottenuti con la la funzione φ_1, sia per quelli ottenuti con la funzione φ_2. In entrambi i casi l'analisi statistica (Figura 11 e Figura 12) ha evidenziato che la tecnica che ha portato a risultati migliori è PSO. Dati i risultati si potrebbe, in futuro, studiare altre tipologie di vicinato ed altre varianti per il controllo della velocità delle particelle, per verificare se si possa giungere a risultati migliori. Inoltre, sarebbe opportuno fare anche una indagine su altre combinazioni degli iperparametri, in particolare sui coefficienti di accelerazione delle particelle.

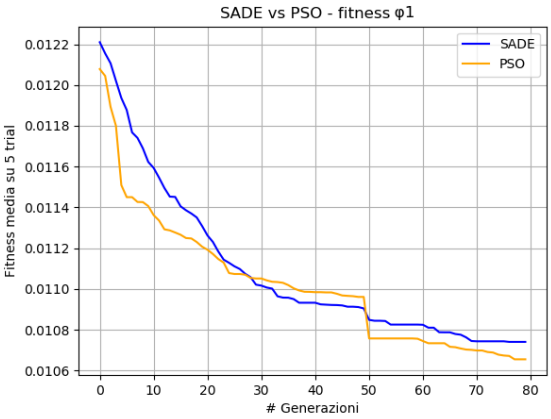

Fig. 11: Confronto PSO e SADE per φ_1.

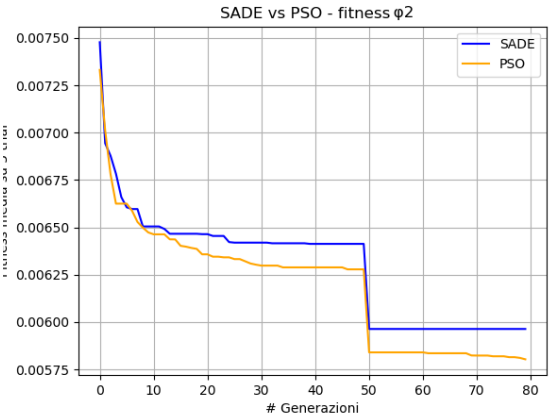

Fig. 12: Confronto PSO e SADE per φ_2.

F. *Confronto delle prestazioni ottenute*

Sono stati, inoltre, messi a confronto i tempi sul giro sui due circuiti utilizzati per l'addestramento, sia per PSO che per SADE, con φ_1 e φ_2, ma considerando soltanto le varianti migliori per ogni tecnica. Le tabelle VI e VII mostrano i miglioramenti che si hanno su ogni giro ogni 5 generazioni. Ogni tempo sul giro rappresenta la media dei tempi dei 5 trial.

	SADE fitness 1 - variante 9				SADE fitness 2 - variante 8			
	CG-Speedway n.1		Forza		CG-Speedway n.1		Forza	
GENS	Lap 1	Lap 2	Lap 1	Lap 2	Lap 1	Lap 2	Lap 1	Lap 2
1	51,70	48,10	116,20	108,57	52,80	48,75	121,84	114,62
5	49,97	45,99	112,71	106,33	50,47	46,57	114,20	107,42
10	48,80	44,72	108,94	101,59	50,08	46,28	111,88	104,92
15	48,53	44,34	107,29	99,88	50,09	46,29	111,92	104,62
20	48,28	43,94	104,64	97,54	49,99	46,11	112,06	104,82
25	47,54	43,54	103,90	96,98	49,73	45,99	111,12	104,08
30	47,38	43,21	102,61	95,73	49,48	45,65	110,42	103,20
35	47,23	42,92	102,58	94,91	49,53	45,69	110,30	103,26
40	47,26	42,94	102,10	94,86	49,53	45,72	110,20	102,86
45	47,18	42,89	101,89	94,78	49,53	45,72	110,20	102,86
50	47,06	43,09	101,66	94,45	49,53	45,72	110,20	102,86
51	46,32	42,49	102,60	95,07	48,34	44,29	105,30	98,41
55	46,42	42,52	101,58	94,75	48,34	44,29	105,30	98,41
60	46,42	42,52	101,58	94,75	48,34	44,29	105,30	98,41
65	46,42	42,12	101,72	94,83	48,34	44,29	105,30	98,41
70	46,09	42,11	101,18	94,17	48,34	44,29	105,30	98,41
75	45,96	42,23	101,26	94,07	48,34	44,29	105,30	98,41
80	45,98	42,26	101,28	93,89	48,34	44,29	105,30	98,41

TABLE VI: Tempi sul giro del controllore durante l'evoluzione con SADE.

	PSO fitness 1 - Local Best				PSO fitness 2 - Global Best			
	CG-Speedway n.1		Forza		CG-Speedway n.1		Forza	
GENS	Lap 1	Lap 2	Lap 1	Lap 2	Lap 1	Lap 2	Lap 1	Lap 2
1	50,85	47,41	112,70	106,56	52,81	48,90	117,38	110,24
5	49,41	45,81	111,10	104,76	51,44	47,14	115,16	107,96
10	49,38	45,63	109,14	101,39	50,52	46,46	111,82	104,48
15	49,22	45,29	107,46	101,41	50,53	46,47	111,02	103,52
20	48,88	44,85	106,86	99,78	50,03	45,98	109,32	102,25
25	48,86	44,84	106,80	99,61	50,01	46,15	109,38	102,19
30	48,56	44,99	106,18	99,28	50,00	45,90	109,06	101,75
35	48,54	44,99	106,18	99,22	49,86	45,80	108,50	101,41
40	48,58	44,92	106,06	99,16	49,86	45,80	108,50	101,41
45	48,53	44,94	106,00	99,09	49,86	45,80	108,50	101,41
50	48,47	44,80	106,26	99,15	49,88	45,73	108,40	101,27
51	46,42	42,43	100,42	93,41	48,70	44,38	104,80	97,36
55	46,42	42,43	100,42	93,41	48,62	44,32	104,76	97,41
60	46,24	42,42	100,82	93,70	48,62	44,32	104,76	97,41
65	46,24	42,14	100,79	93,65	48,62	44,32	104,76	97,41
70	46,11	42,07	100,31	93,38	48,54	44,24	104,46	97,02
75	45,76	41,74	100,76	93,82	48,54	44,24	104,46	97,02
80	45,68	41,62	100,50	93,89	48,54	44,24	104,46	97,02

TABLE VII: Tempi sul giro del controllore durante l'evoluzione con PSO.

Come si può notare, sul primo giro il tempo impiegato è sempre più elevato perché la macchina parte da ferma e la prestazione sul giro è leggermente diversa; dal secondo giro in poi i tempi e le azioni intraprese dal controllore sono sempre gli stessi. Inoltre, è possibile notare che, a prescindere dalla funzione obiettivo e dalla variante, i tempi più veloci si ottengono con l'algoritmo evolutivo PSO, come emerso anche dall'analisi statistica. In media però entrambi gli algoritmi recuperano nelle prime 50 generazioni diversi secondi su CGS-1 e molti più secondi su Forza. Nelle ultime 30 generazioni, che rappresentano la seconda evoluzione, i tempi in SADE con la variante rand-to-best/1/bin non cambiano, mentre nell'altra variante di SADE e in PSO, seppur di poco, i tempi continuano a migliorare.

G. Confronto tra CarSim, PSO e SADE

In Tabella VIII è possibile vedere i notevoli miglioramenti che le due tecniche hanno apportato su CarSim. Su *CG-Speedway n.1* vi è un recupero di circa 7 secondi sul tempo totale, con un notevole incremento anche della velocità di $+10km/h$. I miglioramenti sono molto più evidenti in *Forza* dove sia SADE che PSO recuperano più di 90 secondi con un incremento della velocità di $+15km/h$. In quest'ultimo circuito è possibile notare come i controllori evoluti riescano a stare perfettamente in pista rispetto a CarSim che, invece, passava buona parte della gara sullo sterrato.

Controllore	Lap 1	Lap 2	TOT	Off Track	Top Speed	Circuito
CarSim	52,51	46,02	98,530	0,88	226	CGS 1
PSO	47,886	43,698	91,584	0,72	238,05	
SADE	48,346	44,296	92,642	0,86	236,76	
CarSim	122,22	170,22	292,470	83,1	276	FORZA
PSO	103,168	95,646	198,814	0,82	291,062	
SADE	105,368	98,416	203,784	0,76	291,94	

TABLE VIII: Confronto tra i tempi di CarSim, sui circuiti addestrati, e i tempi dei controllori evoluti con PSO e SADE con φ_2.

Tali risultati sono molto più evidenti se andiamo ad analizzare i profili di posizione e velocità sulle diverse piste. Per quanto riguarda i profili di posizione (Figura 13 e Figura 14) in CG Speedway n.1 si può notare come sia PSO che SADE stiano meno sui bordi della pista, toccandoli solo in curva per via della più alta velocità, esibendo comunque una guida più aggressiva senza repentini aggiustamenti della posizione. Da ciò si evince che i controllori evoluti hanno una tenuta di strada maggiore, garantendo di spingere di più sull'acceleratore senza perdere il controllo. I miglioramenti sulla tenuta di strada sono molto più evidenti in Forza (Figura 15 e Figura 16) dove i controllori evoluti non escono mai di pista, neanche nella curva più ostica del tracciato (i due picchi descritti da CarSim). In tale curva CarSim, arrivando con la velocità acquisita nel precedente rettilineo, sterza ma perde il controllo e va per la tangente finendo sullo sterrato. Al contrario, gli altri due controllori riconoscono la curva e frenano prontamente per poi recuperare in velocità subito dopo spingendo al massimo.

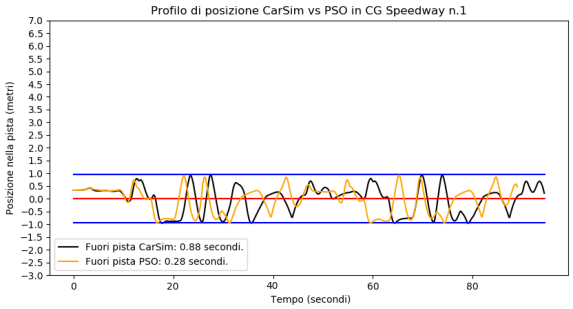

Fig. 13: Posizione in CG Speedway n.1 tra PSO e CarSim.

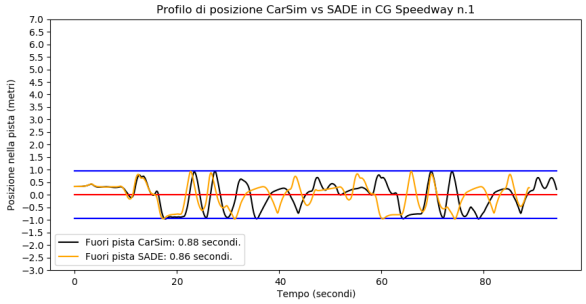

Fig. 14: Posizione in CG Speedway n.1 tra SADE e CarSim.

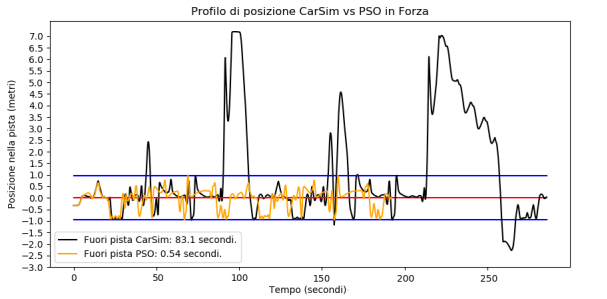

Fig. 15: Posizione in Forza tra PSO e CarSim.

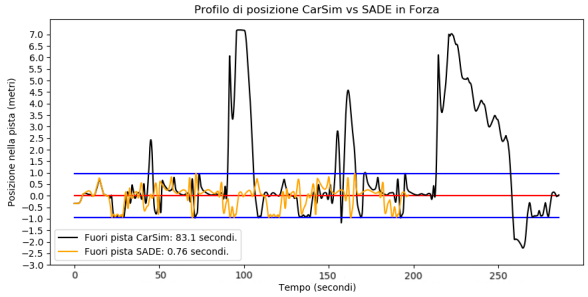

Fig. 16: Posizione in Forza tra SADE e CarSim.

L'andamento nella pista osservato nei profili di posizione sopra riportati è dovuto anche ad una migliore gestione del cambio di marcia, nonché del freno e dell'acceleratore: i controllori raggiungono velocità nettamente superiori e riescono a tenere una buona velocità anche in curva dove, invece, CarSim aveva la tendenza ad inchiodare o uscire fuori pista. Sia in CG Speedway n.1 (Figura 17) che in Forza (Figura 18), PSO ha una partenza più rapida e guadagna decimi di secondo preziosi che lo portano a concludere la gara prima di SADE (in Forza addirittura 5 secondi in meno). Ciò dimostra che il controllore evoluto con PSO, in media, ha un comportamento migliore in curva; al contrario, il controllore evoluto con SADE dimostra una guida a tratti più veloce che lo portano a recuperare sul miglior andamento in curva di PSO.

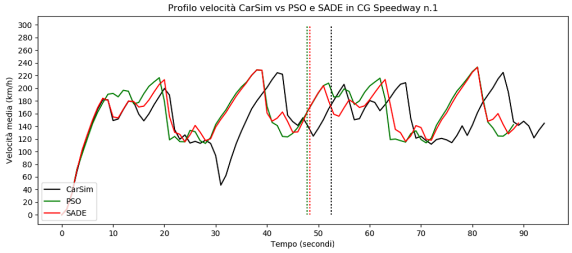

Fig. 17: Profili di velocità media a confronto in CG Speedway n.1. Le linee tratteggiate rappresentano la fine del primo giro per il corrispondente controllore.

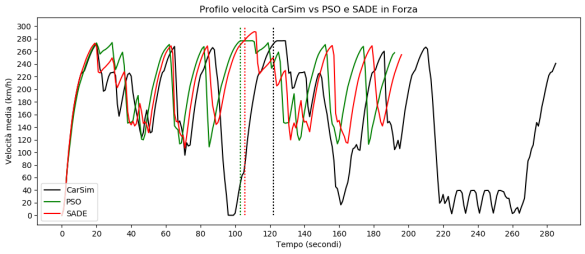

Fig. 18: Profili di velocità media a confronto in Forza. Le linee tratteggiate rappresentano la fine del primo giro per il corrispondente controllore.

H. Risultati su circuiti di Test

Infine, sono stati condotti dei test per verificare che il controllore non si fosse specializzato troppo sui circuiti su cui è stato addestrato. Tali circuiti di test sono un misto tra circuiti veloci (*CG Track 2* e *E-Track 6*) e circuiti che presentano curve senz'altro più complicate (*Alpine 2*, *Street 1*). Come si evince dalla Tabella IX, le prestazioni sono sempre migliori rispetto al driver di partenza CarSim, ma non sono ottimali. Tuttavia, rappresentano comunque un punto di partenza per sviluppare un controllore quanto più generico possibile.

Controllore	Lap 1	Lap 2	TOT	Off Track	Top Speed	Circuito
CarSim	126,048	107,56	233,608	43,28	272,63	STREET 1
PSO	112,232	96,474	208,706	27,26	268,73	
SADE	107,392	124,568	231,960	36,3	270,21	
CarSim	71,202	65,172	136,374	1,92	257,16	CG TRACK 2
PSO	68,694	63,622	132,316	1,58	251,76	
SADE	68,914	63,504	132,418	1,1	249,52	
CarSim	118,678	114,206	232,884	43,9	231,75	ALPINE 2
PSO	109,526	105,416	214,942	28,3	232,35	
SADE	116,5	113,478	229,978	40,32	230,26	
CarSim	110,758	122,272	233,03	25,22	246,08	E-TRACK 6
PSO	106,578	100,758	207,336	13	239,38	
SADE	109,064	104,66	213,724	12,44	240,038	

TABLE IX: Confronto tra i tempi dei controllori su circuiti mai visti.

VIII. Conclusioni e sviluppi futuri

Le conclusioni degli esperimenti condotti mostrano come un approccio evolutivo possa condurre a dei grossi miglioramenti e ottenere, nel caso specifico, prestazioni dell'auto nettamente migliori a quelle di partenza. Quindi, è possibile dire che le due tecniche utilizzate sono efficaci nell'ottimizzazione di un driver virtuale.

Malgrado la poca disponibilità di tempo, il numero di addestramenti, di generazioni e di individui scelti ha portato buoni risultati, ma non si esclude la possibilità di continuare il lavoro variando il numero di generazioni e il numero di individui presenti nella popolazione.

Tra i possibili sviluppi futuri si potrebbe pensare di fare una analisi molto più dettagliata dei parametri per fissare tutti quelli che hanno un impatto minore sulla guida. In tal modo sarebbe possibile ridurre la dimensionalità del problema e considerare un numero di trial più consistente in modo da avere una analisi statistica più precisa. Da come si è notato, i limiti delle prestazioni potrebbero risiedere anche nello stesso codice della funzione *drive()* e su come i parametri siano utilizzati. Ragion per cui potrebbe essere interessante

comprendere a pieno i parametri e provare delle variazioni affinché l'auto raggiunga prestazioni migliori.
Un altro possibile sviluppo futuro potrebbe riguardare l'uso di altre tecniche quali: *Ant Colony* e *Reti Neuroniche* con *Reinforcement Learning*, ed analizzare anche l'impatto in termini di prestazioni e tempi di esecuzione che potrebbero avere rispetto alle tecniche di computazione naturale utilizzate.

REFERENCES

[1] Il sito di TORCS: http://torcs.sourceforge.net/
[2] Il sito di STAC: http://tec.citius.usc.es/stac/ranking_multi.html
[3] La documentazione di Pygmo utilizzata per l'implementazione degli algoritmi: https://esa.github.io/pygmo/documentation/index.html