1. 老師同學好 我是林政旭 我今天要報告的論文是how…
2. 這是我的outline 一開始會先介紹這篇論文想要表達的一些基本背景知識 再來會描述一下問題的假設情境 接著會著重介紹這篇論文提出的modified的演算法 MCPSO 接下來會說明模擬的場景 最後會分別說明論文中提到的兩個實驗 最後再做個總結
3. 這篇論文說在multivehicle systems中task allocation 是最重要的部分，因為他已經被證明是一個np-hard的問題了 所以要完美解決其實並不容易， 在這篇論文主要是在針對救援的方面進行研究 所以會需要考慮無人機載食物或是藥物給受困者的時間 通常這個multivehicle task allocation problems都是用distributed algorithm來解決 像是CBBA, PI, Pi with softmax. 那在這篇論文中，他想要證明centralized algorithm的效率是比distributed algorithm還要來的好的 因為PSO的整體效果比GA還要來的好 跑的也比GA還快一些 也沒有需要微分來去產生下一代，以及PSO程式叫好執行的這些條件，所以作者選擇改良PSO並把它取名為MCPSO
4. 接著就來說明一下問題的假設情境 這裡會有很多種不同type的rescue 的任務 可能像是需要食物、需要藥物 那右上角這些就是無人機 這演算法的目的就是透過給這些無人機分配任務 且這些任務不能互相有衝突 那因為需要透過計算距離 所以這篇論文就定義了一個minimum 的cost function 長的這個樣子 那其中ai 表示第i台車的task list, 那  表示在ai這個list裡面 第k個任務所花費的時間
5. 第一個部分是basic PSO老師跟同學已經講了很多次關於PSO的基本概念了 我就不過多的講解 那w就是銓重 c1 c2 就是固定的值 r1 r2 就是random
   1. Pij是指local best solution的list, pgj是者global best solution 的list
6. 第二個部分是priority determination 就是在決定vehicles去那些地點的順序 每個rescue都有一個end time 來表示最晚可以甚麼時候去 travel time 表示從vehicle現在的位置到rescue的位置的時間 這邊演算法 n是指車子的數量 m是指任務的數量 就是對每一台車子 都需要每個任務的priority 計算距離 時間 把endtime-time去算出difference time 如果diftime 大於0 表示這是可能達成的task 把它放到Set1裡面 如果小於0 就放到set2裡面 這邊應該是用聯集而不是交集 因為交集會把Set清空 接下來就是把set1 set2都order後把他聯集起來
7. 在Particle coding and decoding method中，每個particle代表一種解，每種解都必須把所有tasks都包含進去，所以particle的長度就是task的長度，那就是m個，那這篇論文使用了integer method來決定rescue sequence，其中，u表示有幾種type的任務，所以第一項就是每個type有多少個tasks，他這邊取上高思，自己說完，
8. 接下來就是lbest and gbest solution update methods, current Particle表示particle現在的位置 fail()這個function表示任務failure的數量 也就是說前面這個是指現在位置失敗的任務數量比local best還少 那就更新，那下面這邊這個f()是指time cost 也就是說 如果失敗的任務數量一樣但是時間花費比較少 那一樣要更新 其他情況就維持原樣 如果gbest的失敗任務數量比lbest還要多的話 就更新 如果失敗任務一樣多但花的時間比較少 也要更新 其他狀況就維持不變
9. Particle velocity and location update method, 下面這是basic PSO的function 可以看到上面下面基本上都一樣 就差在更新x位置的時候 這邊多取了上高思 意思就是位置都是整數 這可能跟剛剛decoding的integer method有關 才會這樣設計
10. 下個部分是insert operation，因為PSO是個隨機的演算法 所以可能會出現有車子沒有任務的情況 作者提出了一個解決的辦法 第一步是先計算每台車子任務的數量 接著去看有沒有車子的數量是比平均還要低的 如果有 就把最多任務的那台車的最後一個任務給比較少任務的車子 重複這兩個步驟直接每台車的任務都是m/n個 這個圖就是個例子 總共有13個任務 5台車 平均每台是2.6個任務 他是沒有寫但是我認為應該是平均取下高斯 所以就是每台車最少要2個任務 那第2台車沒有任務 然後第3台車友5個 就把最後一個任務給第2台車 給完之後還是沒有到平均 所以就在重複一次步驟 直到每台車都有2個任務為止 那這是另一個例子 一樣是第2台車沒有任務 最多任務的有第3台跟第5台 就先把一個任務給他 還不夠 所以又再去找最多任務的那台車 把最後一個任務給他
11. 接下來是local search method, 第一個使用了2-opt method, 它是藉由改變任務順序來improve solution的，且它的時間複雜度是O(n^2)，但是因為task 已經藉由priority sort過了，所以changes理應發生在2個不同的type的車子會比較有效果，但是這篇論文只能在同一type間做task 交換，這邊可能是可以再去進步的地方，那這邊u是有幾種type的車，ni是有幾台車，[a,b]是選兩個隨機的車，ma是a車的任務數量，mb是b車的任務數量，接著就是a車的任務第c個任務跟b車的第d個任務交換 然後兩車都重新sort一遍，如果time cost就是fitness value有進步就修改成新的list，如果沒有就照舊
12. 那第二個是exchange operation，它只能執行在the same type of vehicles上，它是改變整個list 而不是單一一個task 這邊u是車子type的數量，一樣是在第i個type中選出2台車，把整個list做交換，重新sort一遍，再去確認fitness value的大小 如果有比較小 就更改 如果沒有 就照舊
13. 這邊就是整個local search method，把剛剛那兩個method都放進來這裡面，他這邊就是有設定一個maxiter1跟maxiter2，如果在for迴圈裏面有更改的話 就break出去 這裡也是一樣 有更新的話就break出去，一開始就是2-opt method 改變兩台車的task 如果有更新就break，沒有就繼續跑下面exchange method 改整個車子的task list，
14. 這個部分是local optimum avoidance，因為PSO演算法有可能會陷入local minimum裡面 所以作者就提出一個方法來解決 第一步是去比較一個例子中現在的global best solution跟上一個的global best solution 然後set threshold的值 如果這兩個global solution的差值比threshold小 count就++ 如果count超過另一個threshold的值 那麼就重新設定起始值 然後重跑
15. 接下來是overview of simulation configurations 會來比較MCPSO 跟CBBA 跟PI 跟PI with softmax 還有GA跟PSO在實驗中的效果 接下來會分別介紹兩個實驗
16. 那第一個實驗分別用4,6,8,10,12,14,16台車 任務的數量剛好是車子的兩倍 且UAV跟helicopter的數量是一樣的 UAV是supply food, helicopter是supply medicine x,y的範圍在-5000到5000裡面 z是在0~1000內 helicopter速度是30，UAV是50，the time window是指截止的時間 如果還有任務在執行中 就會直接取消而且是被計算到failure task裡面 傳送藥品的時間300s 傳食物要350秒 PSO中的參數 權重是0.5+(random/2) 表示權重是在0.5到1之間 c1,c2都是1.494
17. 這個圖是在9次任務中失敗任務的次數 可以看到MCPSO的失敗任務次數都是最少的 那GA跟PSO表現的都沒有很好 失敗次數都很多
18. 這個圖是average fitness function的圖 在7個場景中 MCPSO提供了6個最佳解 PSO提供了1個 這個MCPSO的表現其實也不差 也就比PSO曼一點點而已
19. 這個表是CPU跑的時間 可以看到PI有6個最快 CBBA有1個最快
20. 總得來說 CBBA跑得很快 但是failure rate很高 只能解出13個problem且沒有一個是最佳解，PI跑的幾乎是最快的 且failure rate也很低 但是fitness value是在中間的名次 提供不了最佳解，PI with softmax比PI解出更多的問題 但是也沒有辦法提供最佳解 而且他也是CPU跑得最久，PSO解出了18個 且提供了3個最佳解 但是PSO無法解決複雜的問題 像是超過10 vehicles、20 tasks的話，PSO就無法解決，而GA只解決了7個問題且沒有最佳解而且CPU時間也比PSO還長一點，這就是為什麼選擇了modified PSO而不是GA，最後是MCPSO只有4個問題沒有解出來 而且在59個問題中提供了56個最佳解 雖然CPU時間會隨著場景複雜度而提升 但是是還在可接受的範圍內
21. 接下來是第二個實驗，8台車去分別執行20,30,40,60,80,100個tasks，x,y的範圍改成-10000~10000，z的範圍改成0~1000，the time window 改成20000秒，由於上次實驗的GA,PSO,跟CBBA的表現太差，所以這次實驗只比較MCPSO,PI跟PI with softmax
22. 這張圖可以看到PI的failure number是最多的 MCPSO的failure number是最少的 且只有在tasks是60的時候 MCPSO表現得比較差一點 其餘都比另外兩種演算法表現的好
23. 這張圖可以看到PI的fitness value是最大的 而MCPSO是最小的
24. 這個表格可以看到 PI with softmax 的CPU使用時間是最長的 那在tasks小的時候 PI的CPU使用時間是最短的 但當tasks大於80的時候 MCPSO的CPU使用時間已經比PI還要來的短了
25. Result 自己說
26. 最後是conclusion的部分，在這篇論文中，MCPSO可以解決大部分的問題，即使是比較複雜的問題，MCPSO的表現都還是比PI、PI with softmax要來的好 這兩個實驗展示了centralized演算法的表現是比distributed演算法還要來的好 (CBBA、PI、PI with softmax)，MCPSO可以做為一個distributed algorithm進步的基準