MC-RCPSP : Problème d’Ordonnancement de Projet à Ressources Limitées et Critères Multiples

1. Définition générale

Le RCPSP (Resource-Constrained Project Scheduling Problem) est un problème classique en recherche opérationnelle où l'on cherche à planifier un ensemble de tâches d’un projet, tout en respectant :
- Les contraintes de précédence (certaines tâches doivent être terminées avant que d'autres ne commencent).
- Les contraintes de ressources limitées (chaque tâche utilise un certain nombre de ressources disponibles en quantité limitée).
L'objectif principal est généralement de minimiser la durée totale du projet (makespan).

2. Extension vers le MC-RCPSP

Le MC-RCPSP (Multi-Criteria RCPSP) est une extension du RCPSP classique où plusieurs objectifs sont considérés simultanément :
Par exemple :
- Minimiser la durée totale (makespan).
- Réduire le coût global du projet.
- Optimiser l'utilisation des ressources.
- Réduire le nombre de dépassements de ressources ou équilibrer leur utilisation.
Cela le rend plus proche des projets réels, où plusieurs aspects doivent être optimisés ensemble.

3. Objectif

L'objectif du MC-RCPSP est de trouver un calendrier d’exécution des tâches qui respecte les contraintes de précédence et de ressources, tout en optimisant un ou plusieurs objectifs selon les priorités du projet. L’objectif peut être :

- Mono-objectif (makespan minimal).
- Multi-objectifs (makespan + coût + ressources, etc.).

4. Méthodes de résolution

Plusieurs approches peuvent être utilisées pour résoudre le MC-RCPSP :
- Méthodes exactes : programmation linéaire en nombres entiers (ILP), Branch and Bound.
- Méthodes heuristiques : listes de priorité, règles simples.
- Métaheuristiques : algorithmes génétiques, colonies de fourmis, recuit simulé, etc.
- Approches multi-objectifs : NSGA-II, SPEA2, etc.

5. Formulation mathématique simplifiée

Soit :
- N : nombre de tâches
- R : nombre de types de ressources
- d_i : durée de la tâche i
- r_{ik} : ressource k requise par la tâche i
- b_k : capacité disponible de la ressource k
- S_i : date de début de la tâche i
- P_i : ensemble des tâches précédentes de i

Objectif : Min {makespan = $\max(S_i + d_i)$}

Contraintes :
1. Précédence : $S_j \geq S_i + d_i \; \forall i \in N, \forall j \in P_i$
2. Ressources : $\sum r_{ik} \leq b_k \; \forall k \in R$ à tout temps t

6. Exemple pratique

Prenons l'exemple suivant avec 3 tâches et des ressources limitées :

- Tâches :
  - A : durée 2, ressource 2, coût 5
  - B : durée 3, ressource 3, coût 4 (dépend de A)
  - C : durée 1, ressource 1, coût 3 (dépend de A)

- Ressources totales disponibles : 4 unités

Planification :
1. A commence à t=0.
2. B et C peuvent commencer après A.
3. B utilise plus de ressources et prend donc plus de temps à être planifiée.

7. Exemple de code en Python

```
class Task:
    def __init__(self, name, duration, resource, cost, predecessors=[]):
        self.name = name
        self.duration = duration
        self.resource = resource
        self.cost = cost
        self.predecessors = predecessors
        self.start = None
        self.end = None

def schedule(tasks, max_resource):
    time = 0
    scheduled = []
    while len(scheduled) < len(tasks):
        ready = [t for t in tasks if t not in scheduled and all(p in [s.name for s in
scheduled] for p in t.predecessors)]
        ready.sort(key=lambda x: x.cost)
        for t in ready:
            if t.resource <= max_resource:
                t.start = time
                t.end = time + t.duration
                scheduled.append(t)
                time = t.end
                break
    makespan = max(t.end for t in scheduled)
    total_cost = sum(t.cost * t.duration for t in scheduled)
```

```
    return makespan, total_cost
```

8. Exemple de code en C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

struct Task {
    string name;
    int duration, resource, cost;
    vector<string> predecessors;
    int start = -1, end = -1;
};

bool ready(const Task& task, const vector<string>& done) {
    for (auto& p : task.predecessors)
        if (find(done.begin(), done.end(), p) == done.end()) return false;
    return true;
}

int main() {
    vector<Task> tasks = {
        {"A", 2, 2, 5, {}},
        {"B", 3, 3, 4, {"A"}},
        {"C", 1, 1, 3, {"A"}}
    };
    int time = 0, max_resource = 4;
    vector<string> done;
    while (done.size() < tasks.size()) {
        for (Task& t : tasks) {
            if (t.start == -1 && ready(t, done) && t.resource <= max_resource) {
                t.start = time;
                t.end = time + t.duration;
```

```
            done.push_back(t.name);
            time = t.end;
            break;
        }
      }
    }
    return 0;
}
```

9. Conclusion

Le MC-RCPSP est un problème complexe mais très réaliste pour la planification de projets avec des contraintes réelles. Les exemples fournis ici sont simplifiés, mais ils reflètent bien la logique de base. Des outils avancés permettent d'aborder des projets à grande échelle avec de multiples objectifs à équilibrer.